Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

NJIT New Jersey's Science & Technology University

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road. Ann Arbor. MI 48106-1346 USA
313/761-4700 800/521-0600

UMI Number: 9635206

Copyright 1996 by
Kim, Tea-Quin

All rights reserved.

UMI Microform 9635206
Copyright 1996, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized
copying under Title 17, United States Code.

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

ABSTRACT

GLOBAL FUZZY CONTROL BASED ON CELL MAPPING

by
Tea-Quin Kim

Cell mapping methods can generate global optimal controls for nonlinear dynamic systems. However, the implementation of this approach for on-line control is difficult due to its use of a table-based controller which requires a large amount of memory.

In this dissertation, a cell mapping based systematic method is developed to construct a fuzzy controller to replace the table-based controller for global optimal control of dynamic systems. The method consists of four steps: 1) An optimal control table is generated using a cell mapping algorithm. 2) The cells are linked into trajectories to characterize the optimal dynamic evolution of various cell. 3) The trajectories are classified into groups of similar trajectories so as to represent global optimal controls for the entire cell space with a number of representative trajectory groups. 4) Based on the input-output relations of these trajectories groups, which represent the states and controls of the corresponding cells, a set of fuzzy rules are generated for a fuzzy controller.

With the method of grouping trajectories developed in this dissertation, controls for the entire space can be expressed corresponding to trajectory groups instead of cells with a reduced number of trajectory groups. This reduces the complexity of constructing the global fuzzy controller compared with developing rules based on the control data of all the cells. The developed method is applied to the ship navigation and car parking problems to show the applicability of the method to two and three dimensional problems.

GLOBAL FUZZY CONTROL BASED ON CELL MAPPING

by
Tea-Quin Kim

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Mechanical Engineering

May 1996

Copyright © 1996 by Tea-Quin Kim
ALL RIGHTS RESERVED

APPROVAL PAGE

GLOBAL FUZZY CONTROL BASED ON CELL MAPPING

Tea-Quin Kim

Dr. Ming C. Leu, Dissertation Advisor — Date
Professor of Mechanical Engineering and State Chair in
Manufacturing/Productivity, NJIT

Dr. Bernard Friedland, Committee Member — Date
Distinguished Professor of Electrical Engineering, NJIT

Dr. Denis Blackmore, Committee Member — Date
Professor of Mathematics, NJIT

Dr. Nouri Levy, Committee Member — Date
Associate Professor of Mechanical Engineering, NJIT

Dr. Zhiming Ji, Committee Member — Date
Assistant Professor of Mechanical Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Tea-Quin Kim

Degree: Doctor of Philosophy

Date: May 1996

Undergraduate and Graduate Education:

- Doctor of Philosophy in Mechanical Engineering,
  New Jersey Institute of Technology,
  Newark, New Jersey, 1996

- Master of Science in Mechanical Engineering,
  New Jersey Institute of Technology,
  Newark, New Jersey, 1989

- Bachelor of Science in Mechanical Engineering,
  Korea University,
  Seoul, Republic of Korea, 1985

Major: Mechanical Engineering

Presentations and Publications:

Kim, Tea-Quin, "*Car Parking Control System Using Fuzzy and Cell Mapping Approach,*" Published and presented in Hoboken, New Jersey, April 30, 1994, at the Fifth KSEA Northeast Regional Conference.

Kim, Tea-Quin, "*Analysis and Simulation of Fuzzy Truck-Trailing Backup Control System,*" Published and presented in Polytechnic University, Brooklyn, New York, April 16, 1993, at the Third ASME Regional II Graduate Student Technical Conference.

Kim, Tea-Quin, Ming. C, Leu, Z, Ji, and F.Y. Shih,"*Robot Vision System For 3-D Metrology,*" Published and presented in Pensylvania State University, University Park, Pennsylvania, May 23, 1990, at the NAMRC of SME Conference.

Kim, Tea-Quin, *Robot Vision System For Determining 3-D Coordinates of Object Points,* Master Thesis, New Jersey Institute of Technology, Newark, New Jersey, 1989.

This dissertation is dedicated to
my parents
my lovely wife, HeeKyung Han
my daughters, Sarah and Deborah Kim

ACKNOWLEDGMENT

The author will always remain indebted to his advisor, Professor Ming C. Leu, for his support and patience, and particularly for the confidence he has always expressed in me throughout this research.

Thanks are due to Professors Bernard Friedland, Nouri Levy, Denis Blackmore, and Zhiming Ji for dedicating their time to serve as members of the committee.

The author is grateful to Professors Ronald Kane and Ronyaw Chen for graduate advice and assistance throughout these years.

Timely help and suggestions came from many good friends: Prof. W. H. Zhu, Dr. Byong Bahn, Dr. Hsin-Te Liao, Dr. Hermean Wong, Dr. Shu-Chieh(Roger) Chang, Dr. Z. Deng, and Dr. Naruse.

The author sincerely appreciates SEJONG FOUNDATION in KOREA, which partially supported this work.

The author is grateful to the members of 'The End of The Earth Church', moreover, to Jesus Christ. The author also appreciates Ruth Caputo and Victor Caputo.

The author is grateful to the members of his family for their love, faith, and patience.

Lastly, the author would like to express his greatest and deepest gratitude to his lovely wife, Heekyung (Han) Kim, for the many sacrifices she has made for his education that can not be compensated in any way.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

Cell mapping methods can generate global optimal controls for nonlinear dynamic systems [1, 2]. This approach involves dividing the continuous state space into finite discrete cells and numbering them in order. There are two main features that distinguish this method from others in optimal control studies. First, it performs a global analysis for any given state space, and optimal trajectories for all possible initial states can be determined simultaneously. Second, obstacles in the work space do not increase difficulties in analysis but, instead, lessen the computation burden.

Often, a cell mapping method is used to create an optimal trajectory off-line. In principle, the method can be easily implemented for on-line feedback control by generating a table of optimal controls for all cell states. In the implementation, at each sampling time a system state is measured and the cell corresponding to this state is checked against the optimal control table to look for the control to use in the next time interval. However, since the cell mapping method is an approximate method, the issue of solution accuracy needs to be addressed if the method is to become a viable one for practical closed-loop control applications. Moreover, a problem with table-based control is that the table can be very large, especially if the system has many inputs and outputs. This, in turn, increases the memory requirement, thus affecting the implementation cost of the system. Table-based control often gives a bumpy response as the controller jumps from one table value to another.

The method of cell mapping has been used to solve various engineering problems [3, 4, 5, 6, 12], and it has been incorporated in various studies such as the construction of a fuzzy-neural system [7, 8, 9]. The drawbacks of a cell space optimal controller can be remedied by constructing a fuzzy controller from the cell mapping data. From the perspective of cell mapping, a fuzzy-neural system can provide a mechanism to replace the table-based control, thus improving control smoothness and reducing computer memory and computational burden. From the perspective of a fuzzy-neural system, a cell mapping method is useful as follows:

i) The optimal control strategy obtained from cell mapping can be incorporated into the fuzzy controller. The strategy cannot be extracted by human experts in a systematic way for nonlinear dynamical systems, especially those with the existence of inequality constraints. The optimality is realized by deriving optimal rules and/or modifying parameters of a fuzzy-neural system based on the numerical information of optimal cell mapping.

ii) The global stability of the fuzzy-neural control system can be determined by cell mapping.

Thus, the use of a fuzzy controller constructed from the cell mapping data provides a promising complete solution for the global optimal control of nonlinear systems. However, the design of fuzzy rules to construct fuzzy system is not an easy problem if the amount of optimal cell mapping data is large. A rule of the fuzzy controller is to represent a set of cells which have similar control activities. In this thesis, a systematic method of fuzzy rule generation is proposed.

It is desirable to have a fuzzy controller for the entire cell space with a small number of fuzzy rules. In the worst case, the number of fuzzy rules is the same as the number of cells. This may occur when the control activity of each individual cell is very different from its neighbor cells. To construct a fuzzy controller based on the numerical information of the optimal cell mapping table, there are two kinds of difficulty: first, how to design the rules of the fuzzy controller; second, how to manage the great amount of numerical information generated by cell mapping.

In this study, we develop a systematic method to synthesize a fuzzy controller from the cell mapping based optimal control data. This method is distinguished from others in that it reduces the optimal trajectories in the cell space into a small number of selected optimal trajectories. Optimal trajectories are created from the cell data, and similar trajectories are collected as a group of trajectories. A trajectory is selected to represent each group of trajectories. The selected trajectories are used to generate the fuzzy rules. The developed method is applied to the ship navigation and car parking problem.

In Chapter 2 we provide a general discussion of cell mapping and fuzzy systems. In Chapter 3 we propose a systematic method to analyze the cell mapping data and to construct a fuzzy controller based on the cell data. In Chapter 4 we apply the proposed method to the ship navigation and car parking problems. In Chapter 5 we present our conclusions.

CHAPTER 2

CELL MAPPING AND FUZZY SYSTEM

This chapter will provide a discussion of the basis for constructing a global fuzzy controller based on the numerical information generated from optimal cell mapping. First, a cell mapping algorithm to generate an optimal trajectory (or path) will be discussed. The optimal trajectory includes the corresponding global optimal controls. To generate the global optimal trajectory, the description of the path in task space can be converted to a description in discrete cell state space. Based on the discrete cell state space, the cell mapping algorithm [2, 3] plans the trajectory with the dynamic equations of a system and the constraints of actuators in order to generate feasible paths which allow near optimal motions to be executed. Thus, the cell mapping constructs tabular numerical information which represents trajectories (or paths) and their related control actions corresponding to the discrete state space.

A fuzzy controller [20] is a mechanism that consists of a set of control rules for a fuzzy rule base and processes them with a systematic logic inference. The rules in the fuzzy controller take IF -THEN form with linguistic labels. The fuzzy rule base consists of a collection of rules which represents all possible situations and their corresponding control actions. The systematic logic inference converts the linguistic labels of the rule base to a final crisp control action. The systematic logic inference consists of fuzzification, implication, aggregation, and defuzzification.

2.1 Cell Mapping Algorithm

2.1.1 Formulation of Cell State Space Path Optimization Problem

To generate global optimal trajectories of a robot, it is desirable to divide the continuous state space into finite discrete cells. Then, motion of a robot is transformed into a discrete cell space $\boldsymbol{Z}$. The discrete cells are formed in rectangular shape. To construct the discrete cell space, first the subspace of the continuous space is divided into a number of intervals with size h_i for axis x_i. An interval is denoted by an integer z_i , i.e.

$$\left(z_i - \tfrac{1}{2}\right) h_i \le x_i \le \left(z_i + \tfrac{1}{2}\right) h_i \qquad (2.1)$$

The n-tuple $(z_1,...,z_n)$ is called a cell and is denoted by z. n is the dimension of the cell space; n is 2 for (x, y) and is 3 for (x,y,θ).

In the discrete approximation, the control output from actuators is a finite dimensional vector, **u**, given by:

$$\mathbf{u} = [\, u_1, u_2, ..., u_p] \qquad (2.2)$$

Let the differential equation of an n-dimensional dynamic system be expressed as

$$\dot{x} = f(x, u(t), t) \qquad (2.3)$$

where $x \in R^n, u \in R, and\ t \in R$. By solving this set of differential equations, a discrete point mapping can be established in the form of

$$x(k+1)=x(k) + \int_{t_k}^{t_{k+1}} f(x(k),u(k))dt \qquad k=1,..,N \qquad (2.4)$$

A cell mapping can be constructed from a point mapping and written in the form of

$$z(k+1) = H(z(k), u(k), t(k)) \tag{2.5}$$

where $z \in I^n$ is an n-tuple of integers, $k \in I$ and $u \in U$. The discretization of time is artificially limited to a range, $t(k) \in T$. The total number of cells in the cell state space is N_C.

If the trajectory starting from the representative point of an initial cell ends at a point in another cell after a duration of time, the cell containing the end point is called the *image cell* and the initial cell is called the *domain cell*. Then, the dynamics of the system can be expressed in terms of cell-to-cell mappings instead of point-to-point mappings. Determining the image of a cell can proceed as follows. For a given cell $z(k)$, first find the coordinates of its center $x(k)$. Under control $u(k)$ and time duration $t(k)$, $x(k+1)$ is determined as the image of $x(k)$ by point mapping. If the corresponding cell of point $x(k+1)$ is $z(k+1)$, then $z(k+1)$ is the image cell of $z(k)$.

The functional that is to be minimized for finding a near-optimal path assumes the form

$$J_C = \sum_{k=1}^{m} \left(c_k \int_{t_k}^{t_{k+1}} f_0(x_k, u_k) dt + t_k \right) \tag{2.6}$$

where k is a number used to indicate the mapping step, m is the total step number from the cell $z(k)$ to the target cell, and c_k is an appropriate coefficient used to weight the two

terms in J. For time optimal control, all c_k's are zero. If it is important to save control energy, the c_k's should be large.

2.1.2 Path Plan with a Recursive Depth-First-Backtracking Search Method

After the discrete cell state space is constructed and the goal states (target) are given, the path planner searches the paths from various initial states to the target in the discrete cell state space satisfying all the conditions and dynamic equations in Section 2.1.1. Cell-to-cell mappings are performed to search for optimal paths from various initial states to the target. The original cell-to-cell mapping algorithm developed by Hsu [2] generates all possible paths and then searches optimal paths with a ***systematic search algorithm***. The systematic search algorithm is similar to the breadth-first search algorithm of artificial intelligence [22]. The search algorithms of artificial intelligence are of two kinds: breadth-first and depth-first . They determine the order in which states are examined in a tree or graph. Zhu [3] improved the systematic search algorithm with a series of modifications. The modifications include rearrangement of the processing cells. The search starts near the target cell to reduce the search time.

Zhu and Leu [4] used a depth-first-backtracking search algorithm to solve the robot optimal trajectory planning problem. The depth-first-backtracking search algorithm needs less storage compared to the breadth-first search algorithm since it does not have to keep all the node cells at a given level on the storage list (or **OPEN** list) [22]. Especially, starting near the target cell, the depth-first-backtracking algorithm overcomes the disadvantage of getting 'lost' deep in space.

A depth-first-backtracking search algorithm is used in this study. The pseudo code for the optimal cell mapping is given below.

```
ZDL  ; lists of unprocesses cells
ZUL  ; lists of underprocessing cells
ZPL  ; lists of processed cells.

Initialize: ZPL := [target cells]; ZDL := [1,...,Nc]; % Nc is the total number of cells
while3 ZDL is non-empty do; % Use all cells to the cell mapping

Optimal Cell Mapping Procedure:

Initialize: ZUL := [z]; % z is the starting cell which is selected in several ways
begin;
set i=1;
while2 ZUL is non-empty do; % ZUL = [X, .....]
   begin
   X is the leftmost cell on ZUL and cs(i) are the image cells of X w.r.t control input ui
(i=1,....,k) of equation (2.2)
   if i==k then go to sort;
      else
         if cs(i) is one of the status as following
         case 1: cs(i) is a cell on the list ZPL:
                  It is one of possible paths from X to target via cs(i);
                  i = i+1;
                  Optimal Cell Mapping Procedure % backtrack
         case 2: cs(i) is a cell on the list ZUL;
                  skip; % Mark algorithm manner
                  i = i+1;
                  Optimal Cell Mapping Procedure % backtrack;
         else % unprocessed cell and depth-first search manner
                  add cs(i) to ZUL; % LIFO manner
                  set i = 1;
      end if
   sort: % determine optimal path from X to one of its image cell cs(i), where all the cost
from cs(i) to
         target are determined using Jc of equation (2.6):
         add X to ZPL;
         delete X from ZUL;
         Optimal Cell Mapping Procedure % backtracking manner
   end if
end; % end of while2
end; % end of while3
```

The algorithm uses three lists to keep track of cells in the cell state space. The purpose of using three lists is that the planner never explores a path twice from the same cell, but it records the exact path produced by the search. If the cell is marked as a processed cell, a search will never start from this cell again. Figure 2.1 illustrates the cell mapping algorithm. Each cell has the shape of a rectangular parallelepiped. The two point-to-point mapping trajectories in Fig. 2.1(a) can be represented by a single cell-to-cell mapping trajectory in 2.1(b). With the cell-to-cell mappings, the behavior of the dynamic system can be represented by mappings in cell space.

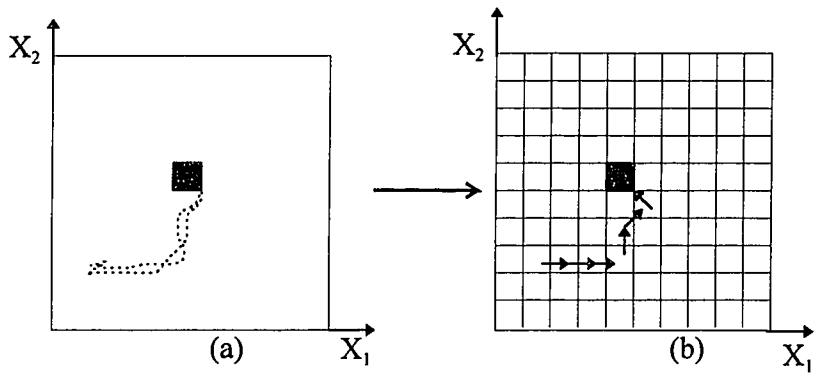


Figure 2.1 (a) Two point-to-point mapping trajectories. (b) The corresponding discrete cell state space and a cell-to-cell mapping trajectory. The black rectangle is the target cell.

2.1.3 Global Information as a Control Table

During the cell-to-cell mapping using the depth-first-backtracking search method in Section 2.1.2, global information is generated in a table which is the ZPL of the pseudo code of Section 2.1.2. The fields of the table are: z_i:(o_i, $\boldsymbol{d}_i$, $\boldsymbol{u}_i$, s_i, p_i, I_i : i=1,..,r), where the symbols in the parenthesis represent, respectively, cell order, coordinates of each cell, discrete control inputs, step numbers to the target cell, performance index, and the image

cell number of a domain cell z_i. If all the cells are controllable cells, then r equals N_C. The bold character represents a vector and the plain character represents a scalar. For example, the coordinates of each cell, $\boldsymbol{d}$, can be represented by (x, y) coordinates. A table-based controller can be implemented with the fields information. In principle, the table-based controller can be easily implemented in on-line feedback. In the implementation, at each sampling time a system state is measured and the cell corresponding to this state is checked against the optimal control table to look for the control to use in the next time interval. However, since the cell mapping method is an approximate method, the issue of solution accuracy needs to be addressed if the method is to be viable for practical closed-loop control applications. Moreover, a problem with table-based control is that the table can be very large, especially if the system has many inputs and outputs. This, in turn, increases the memory requirement, thus affecting the implementation cost of the system. Table based control often gives a bumpy response as the controller jumps from one table value to another.

2.2. Fuzzy Systems

The main goal of this dissertation study is to construct a global fuzzy controller based on the numerical information that is generated using cell mapping. To construct a fuzzy controller, the following issue must be addressed: *fuzzy rule generation.* Fuzzy rule generation is related to partitioning of the input state space for determining the number of rules. In this section, a fuzzy controller uses four main processes which are fuzzification, implication, defuzzification, and aggregation (or inference). Figure 2.2 illustrates the structure of a rule-based fuzzy controller.

2.2.1 Fuzzy Expert Controller

The rule in a fuzzy controller takes the following form:

If <condition>, then <control action> (2.7)

The <condition> is the premise part and the <control action> is the consequence part. For example, a fuzzy control rule to steer a vehicle toward a parking target may be: If **Parking Target** x is '**Ahead,**' then **steering angle** y is **'straight.**" The terms x and y are referred to as the state variable and control variable, respectively. The terms "**Ahead**" and "**straight**" are linguistic labels.

The adjectives such as(ahead, straight) in the rules do not correspond to precisely defined ranges. A parking target that is located some degrees from the straight direction could be described as "straight". A fuzzy controller takes this into account, so the same rule can work for many different situations.

2.2.2 Linguistic Labels and Membership Functions

A fuzzy variable is a linguistic term (or label) associated with some membership function. In Figure 2.3(a), there are three linguistic labels in the x_1 axis; they are F_1^1 , F_1^2, and F_1^3. These three linguistic labels may be interpreted as Negative-Big, Zero, and Positive-Big, respectively. Here the membership functions have trapezoidal shapes. There are many other types of membership functions such as bell shaped membership functions.

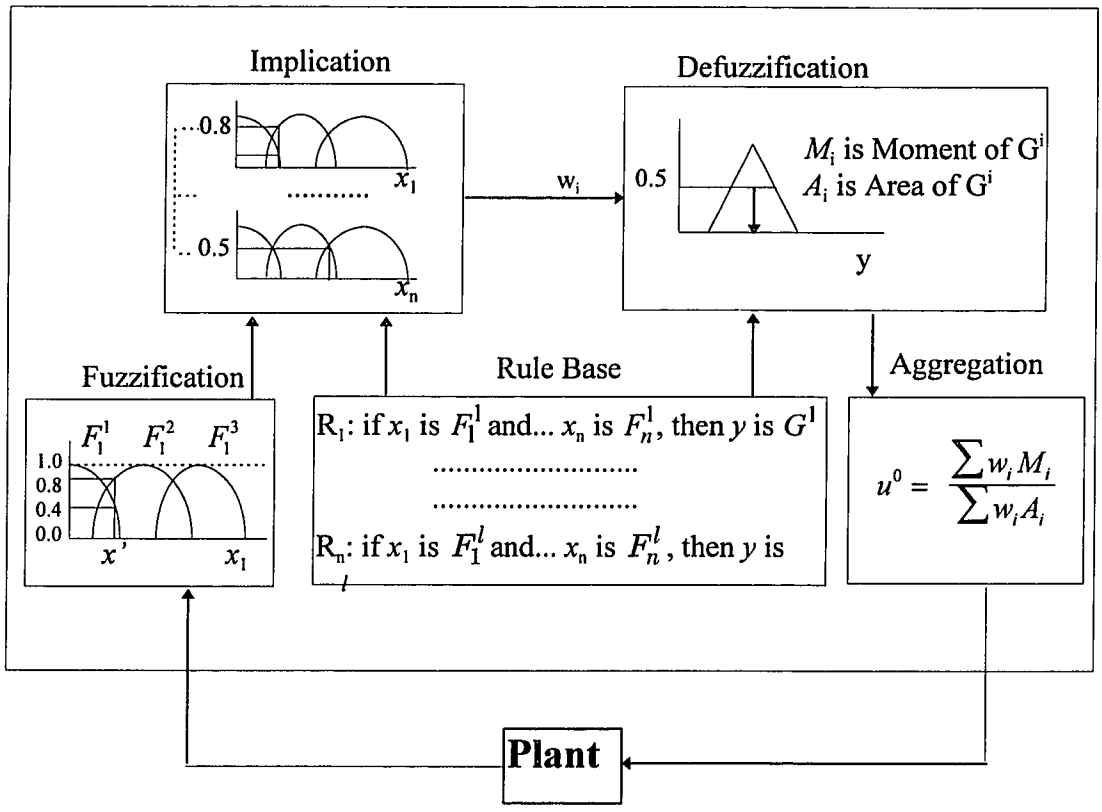
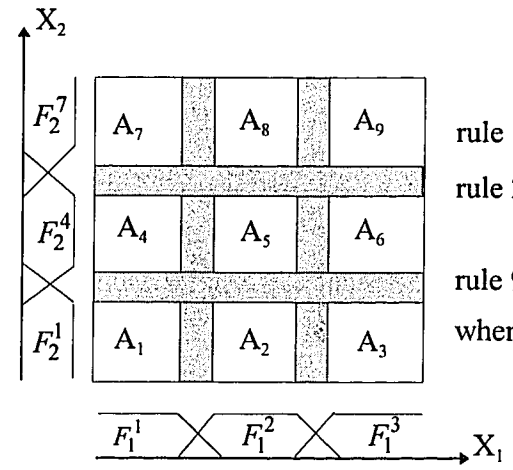


Figure 2.2 Structure of rule-based fuzzy controller



rule 1: If x_1 is F_1^1 and x_2 is F_2^1, then y is A_1

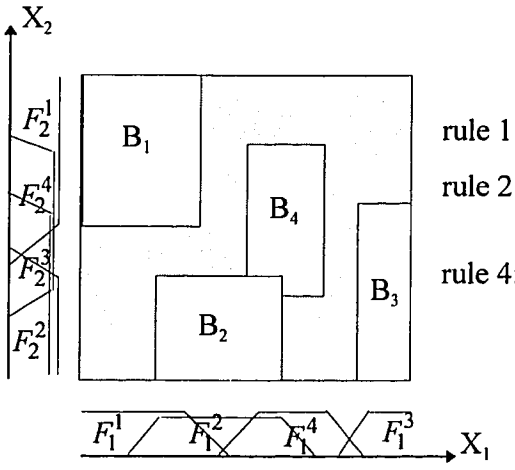
rule 2: If x_1 is F_1^2 and x_2 is F_2^2, then y is A_2

.....

rule 9: If x_1 is F_1^9 and x_2 is F_2^9, then y is A_9

where $F_1^1 = F_1^4 = F_1^7$, $F_2^1 = F_2^2 = F_2^3$, etc.

(a)



rule 1: If x_1 is F_1^1 and x_2 is F_2^1, then y is B_1

rule 2: If x_1 is F_1^2 and x_2 is F_2^2, then y is B_2

.....

rule 4: If x_1 is F_1^4 and x_2 is F_2^4, then y is B_4

(b)

Figure 2.3 (a) Regular input space partition and its related rules (b) clustering input space partition and its related rules.

2.2.3 Rule Base

The fuzzy rule base consists of a collection of fuzzy rules. This represents qualitative knowledge in a fashion which can be interpreted by the rule based fuzzy logic process. Two commonly used rule base representations are as follows. One is the Mamdani [20] type rule base representation:

$$\begin{aligned} &R_1\text{: if } x_1 \text{ is } F_1^1 \text{ and } x_2 \text{ is } F_2^1 \ldots x_n \text{ is } F_n^1 \text{, then } y \text{ is } G^1; \\ &R_2\text{: if } x_1 \text{ is } F_1^2 \text{ and } x_2 \text{ is } F_2^2 \ldots x_n \text{ is } F_n^2 \text{, then } y \text{ is } G^2; \\ &\qquad\qquad \ldots\ldots\ldots\ldots \\ &R_n\text{: if } x_1 \text{ is } F_1^l \text{ and } x_2 \text{ is } F_2^l \ldots x_n \text{ is } F_n^l \text{, then } y \text{ is } G^l; \end{aligned} \tag{2.8}$$

where F_1^i, F_2^i ,..., and G^i are premise and consequence linguistic labels, respectively (or fuzzy sets).

A different way of describing the rule base, which is the Takaki and Sugeno type [10], is to have y as a function of the state variables:

$$\begin{aligned} &R_1\text{: if } x_1 \text{ is } F_1^1 \text{ and } x_2 \text{ is } F_2^1 \ldots x_n \text{ is } F_n^1 \text{, then } y = f_1(x_1,\ldots, x_n); \\ &R_2\text{: if } x_1 \text{ is } F_1^2 \text{ and } x_2 \text{ is } F_2^2 \ldots x_n \text{ is } F_n^2 \text{, then } y = f_2(x_1,\ldots, x_n); \\ &\qquad\qquad \ldots\ldots\ldots\ldots \\ &\qquad\qquad \ldots\ldots\ldots\ldots \\ &R_n\text{: if } x_1 \text{ is } F_1^l \text{ and } x_2 \text{ is } F_2^l \ldots x_n \text{ is } F_n^l \text{, then } y = f_n(x_1,\ldots, x_n); \end{aligned} \tag{2.9}$$

where $\boldsymbol{x} = (x_1,\ldots,x_n)^T \in R^n$ and $y \in R$ are the input and output of the fuzzy system, respectively. The control designer must identify all the premise and consequence linguistic labels. This is often called the system structure identification problem in fuzzy

control researchers. The parameters of the consequence labels can be determined using optimization methods such as Kalman filter and neural-network algorithms [8, 10, 11] with the given input-output numerical data. It is called the parameter identification problem by fuzzy control researchers [16].

The rule base of the Takaki and Sugeno type controller has an advantage over that of the Mandani type controller when control actions can not be determined well by linguistic labels. But a drawback of the Takaki and Sugeno type controller is the need to determine the parameters of the consequence part.

2.2.4 Fuzzification

The process of fuzzification is assigning a brief value to a real-valued input variable with respect to each membership function. The fuzzification maps each coordinates x_i of a crisp point $\boldsymbol{x} = (x_1,...,x_n)^T \in R^n$ into a fuzzy set F_i^j. For example, $x_1 = x'$ in Fig 2.2, the truth values are

$$\begin{aligned} \mu_{F_1^1}(x') &= 0.4 \\ \mu_{F_1^2}(x') &= 0.8 \\ \mu_{F_1^j}(x') &= 0 \text{ otherwise} \end{aligned} \qquad (2.10)$$

2.2.5 Implication

Implication transfers the "brief" values of the premise part to the output of the fuzzy logic system, which is a consequence part. Each fuzzy IF-THEN rule defines a fuzzy implication. Many fuzzy implication rules have been proposed in the fuzzy logic literature. Two commonly used fuzzy implication rules are as follows:

1) Mamdani implication or Min-operation rule of fuzzy implication:

$$\mu_{F_1^i \times \cdots \times F_n^i \to G^i}(\underline{x}, y) = \min\left[\mu_{F_1^i}, \mu_{F_2^i}, \cdots, \mu_{F_n^i}\right] \tag{2.11}$$

2) Product-operation rule of fuzzy implication:

$$\mu_{F_1^i \times \cdots \times F_n^i \to G^i}(\underline{x}, y) = \mu_{F_1^i} \times \mu_{F_2^i} \times \cdots \times \mu_{F_n^i} \tag{2.12}$$

2.2.6 Defuzzification

Defuzzification is important to a fuzzy control system. It generates the real valued control. There are many defuzzification methods [20]. Among them two defuzzification methods, namely the center of gravity method and the mean of maximum method, are used frequently.

2.2.7 Aggregation

Since the multiple rules of a fuzzy control system can be active simultaneously, all of the active rules are combined to create the final result. The fuzzy inference engine performs a mapping based upon the fuzzy IF-THEN rules in the rule base. Let x_1, x_2,..., x_n be the inputs of the fuzzy controller and y be the output of the controller. The truth values w_i (i = 1,..,n), which represent weights of rules when using the Mandani implication, are: w_i = min $[\mu_{F_1^i(x_1)}, \mu_{F_2^i(x_2)}, \ldots, \mu_{F_n^i(x_n)}]$. Suppose the fuzzy controller consists of the fuzzy rule base described by equation 2.8, then one simple way to combine the rules is by taking the weighted average of the outputs:

$$u^0 = \frac{\sum_{i=1}^{n} w_i y^i}{\sum_{i=1}^{n} w_i} \qquad (2.13)$$

where u^0 is the control to the plant. This is called a correlation-minimum inference method.

Fuzzy rules can be constructed by an expert with lots of experiments for the states that the plant will encounter. Another approach is to use a linguistic description, viewed as a fuzzy model of the process under control, to derive the set of rules. Karg and Vachtsevanos [9] generated fuzzy rules based on cell mapping with a tree search approach. In their method, each cell of the cell state space is used as a fuzzy region, i.e. a region for a fuzzy rule. After applying a control strategy, several trajectories are generated. From these trajectories they selected a possibly optimal trajectory to the target cell. Based on the selected trajectory, the fuzzy rules are generated by assigning linguistic control variables to the corresponding control values of the trajectory. Kosko [13] developed a space clustering method to obtain fuzzy rules with the fuzzy associative memory (FAM). Wang and Mendel [14] proposed a fuzzy rule generation method by assigning a degree to each rule from input and output sample data. Lin and Lee [15] constructed the input-ouput mapping using a neural network. Carr [18] proposed the use of a genetic algorithm [23] to determine the membership functions of fuzzy rules for the PH control problem. Abe and Lan [17] used a min-max operator to find the control pattern with a hypercube window. Chang [24] applied fuzzy logic and neural networks to adaptive nonlinear control problems.

In this dissertation study, the fuzzy controller construction is based on cell-to-cell mappings. The main goal of this study is to construct a global fuzzy controller using the numerical information that is generated using cell mapping. The purpose of the fuzzy controller is to achieve smooth motion control response with relatively small computational and memory burden while retaining the optimality of the table-based control. This will be discussed next.

CHAPTER 3

ALGORITHM FOR GROUPING OPTIMAL TRAJECTORIES

The generation of fuzzy rules based on cell-to-cell mappings is a challenging research issue. This chapter will identify the characteristics of the numerical information obtained from cell-to-cell mappings and use them for efficient generation of the fuzzy rules of a fuzzy controller.

We will discuss how to create optimal trajectories and groups of similar optimal trajectories that approximate the global optimal control system with a smaller number of trajectories. To create trajectories, each cell is characterized as an initial cell, a simple cell, or a merged cell. To group similar trajectories, two features are used to check the similarity among trajectories. They are: 1) the location of start and ending cells and 2) the highest accumulated control level. Figure 3.1 illustrates the processes of trajectory grouping.

3.1 Determining Trajectories in a Cell State Space

For the purpose of constructing an optimal fuzzy controller, we are interested in obtaining a finite number of trajectories from the table of optimal control data from the entire cell space, and then grouping them based on similarity features among these trajectories. To obtain these finite trajectories, we introduce the concept of initial cell, simple cell, merged cells.

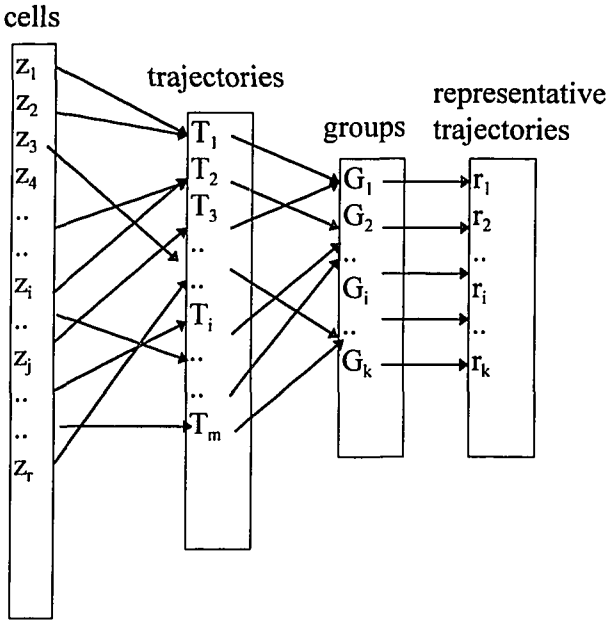


Figure 3.1 Grouping of cells based on similar trajectories

Definition 3.1 : Initial Cell (IC)

A cell z_i is an initial cell if there exists ***no*** z_j such that $H(z_j) = z_i$ and $z_i \neq z_j$ and $j=1,...,N_c$, where H is the cell mapping function of Eq. (2.5) and N_c is the total number of cells in the cell state space.

Definition 3.2 : Simple Cell (SC)

A cell z_i is a simple cell if there exists ***only one*** z_j such that $H(z_j) = z_i$ and $z_i \neq z_j$ and $j=1,...,N_c$

Definition 3.3: Merged Cell (MC)

If there exists ***more than one*** z_j such that $H(z_j) = z_i$ and $z_i \neq z_j$, then z_i is a merged cell.

The gray rectangles, IC_j and IC_k, are initial cells in Figure 3.2. The white rectangles are simple cells. The black rectangles are merged cells. A trajectory is defined as a connection from an initial cell to a merged cell or from a merged cell to another merged cell (MC).

Definition 3.4 : Trajectory

A trajectory is a set of connected cells which evolves either from an initial cell to a merged cell or from a merged cell to another merged cell. The first is the type I and the second is the type II trajectory.

The following procedure is used to derive all the trajectories in a cell state space:

Trajectory Determining Procedure:

Step 1: Each cell in the set $\{z_1, z_2, ..., z_r\}$ is assigned as an IC, SC, or MC based on Definition 3.1 through Definition 3.3.

Step 2: Construct the set of $\Re$:$\{ S_1, S_2,..., S_p\}$ of IC's and MC's.

Step 3: From any $S_i \in \Re$, cells are linked according to Eq. (2.5) until $S_j \in \Re$, $S_i \neq S_j$, is encountered. This forms one trajectory.

Step 4: The **Step 3** process is performed repetitively for every $S_i \in \Re$.

For illustration, there are three trajectories in Figure 3.2: one from IC_j to MC_h, one from IC_k to MC_h, and one from MC_h to MC_p.

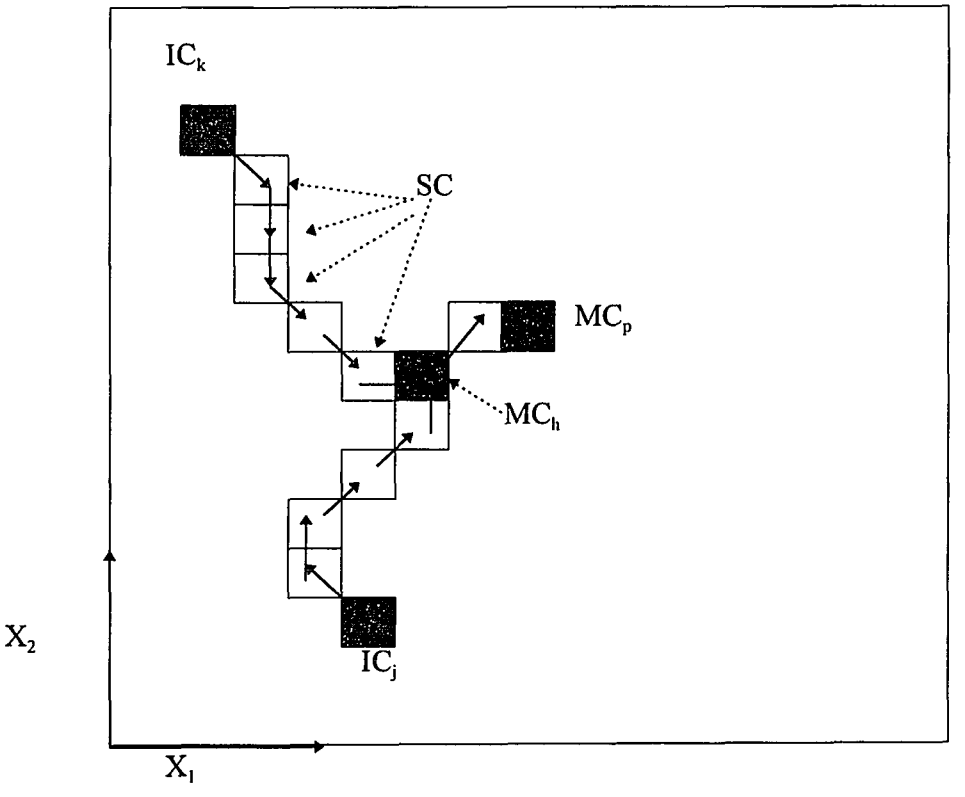


Figure 3.2 Illustration of simple cells, initial cells, merged cells, and trajectories on cell state space.

3.2 Features of Trajectories in a Cell State Space

To group trajectories in a cell state space, the similarity of trajectories should be considered. The similarity of trajectories must be a quantitative measure representing trajectory features. The first task in feature extraction is to determine what features should be used for a trajectory. It is logical to consider features which distinguish trajectories by their locations and control activities.

Since trajectories are formed by connecting cells, the difference between two trajectories can be determined by checking individual cells of trajectories with their locations and control levels. For example, if there are two trajectories, T_a and T_b, representing the following tables of locations and control levels of individual cells. z_a^k, d_a^k and u_a^k are the cell designation, location of the cell, and control level of the cell, respectively, of trajectory T_a. z_b^h, d_b^h and u_b^h are the cell designation, location of the cell, and control level of the cell, respectively, of trajectory T_b.

If the difference between the locations and control levels of the individual cells of these two trajectories is small enough, these two trajectories can be said to be similar. However, comparing the individuals becomes very difficult if $\mathbf{n} \neq \mathbf{m}$ which is the general situation.

Table 3.1: Trajectories T_a and T_b

Trajectory T_a :

z_a^1	z_a^2	.....	z_a^k	.....	z_a^m
d_a^1	d_a^2	.....	d_a^k	.....	d_a^m
u_a^1	u_a^2	.....	u_a^k	.....	u_a^m

Trajectory T_b :

z_b^1	z_b^2	.....	z_b^h	.....	z_b^n
d_b^1	d_b^2	.....	d_b^h	.....	d_b^n
u_b^1	u_b^2	.....	u_b^h	.....	u_b^n

Instead of using all individual cells, we use the select features that are the locations of the starting and ending cells of each trajectory and two highest control levels and the corresponding number of cells with these control levels. The rationale of choosing these features is: 1) The start and ending locations provide an essential indication of the location of a trajectory, and 2) The two highest control levels of a trajectory represent the dominant control levels of the trajectory. It is obvious that even though two trajectories evolve from similar locations of starting cells to similar locations of end cells, the trajectories should be classified as different trajectories if the control levels are very different.

Suppose there are five discrete control levels $\mathbf{u} = [u_1, u_2, u_3, u_4, u_5]$ and all the control levels of T_a in Table 3.1 are u_1, then the control feature of T_a is $\{ u_1, m \}$, where m is the number of cells with u_1. If T_a has control levels u_2 and u_3 with k cells using u_2 and h cells using u_3, then the representation of the control feature is $[\{u_3, h\},\{ u_2, k \}]$ if $h > k$. Here $m = h + k$ if no other control levels are used by any cells. Only the two highest control levels are considered in the control feature because these control levels are dominant in the evolution of the trajectory.

Let the first and second highest control levels of a trajectory T_i be u_i^I and u_i^{II} and the numbers of two control levels be n_i^I and n_i^{II}. Table 3.2 illustrates the features of trajectories T_a and T_b.

Table 3.2 The features of trajectories T_a and T_b

Features of a trajectory T_a :

z_a^l	z_a^f
d_a^l	d_a^f

(a) Location feature

u_a^I	u_a^{II}
n_a^I	n_a^{II}

(b) control feature

Features of a trajectory T_b :

z_b^l	z_b^f
d_b^l	d_b^f

(a) Location feature

u_b^I	u_b^{II}
n_b^I	n_b^{II}

(b) control feature

To summarize, the features of trajectory T_i are: 1) locations of start and end cells, which are z_i^l and z_i^f, and 2)the two highest control levels, which are u_i^I and u_i^{II}, and 3) the numbers of cells with these control levels, which are n_i^I and n_i^{II}.

Based on the features, we can compute the difference of these two trajectories as follows:

$$\eta_1 = \max [\, |d_a^l - d_b^l| , |d_a^f - d_b^f| \,] \tag{3.1}$$

$$\eta_2 = \max [\, |u_a^I - u_b^I| , |u_a^{II} - u_b^{II}| \,] \tag{3.2}$$

$$\eta_3 = \max [\, |n_a^I - n_b^I| , |n_a^{II} - n_b^{II}| \,] \tag{3.3}$$

$$\beta = \gamma_1\eta_1 + \gamma_2\eta_2 + \gamma_3\eta_3 \quad (0 \le \gamma_i \le 1) \tag{3.4}$$

The value of γ_i can be chosen depending on the relative importance of the features.

3.3 Forming Similar Trajectories into a Group

If there are trajectories which have similar locations and similar control activities, then it is possible to combine them into a group of similar trajectories called a trajectory group. The basis for grouping similar trajectories is Equation (3.4): If $\beta < \beta_T$, we say that the two trajectories under consideration are similar.

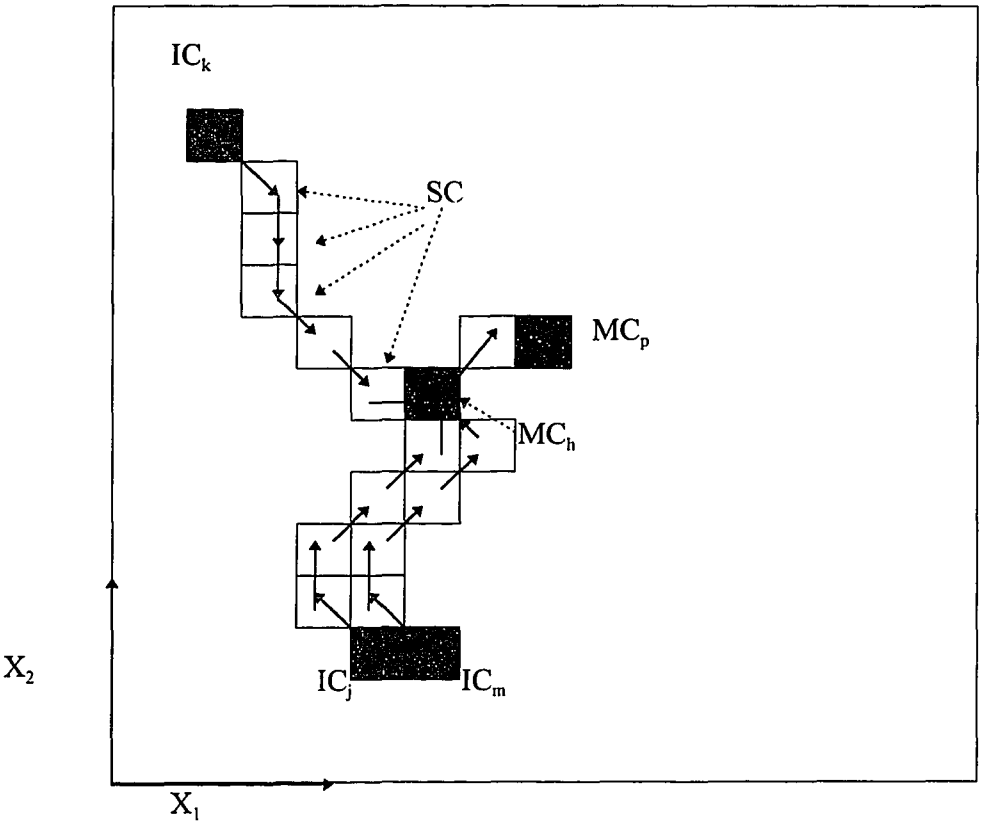


Figure 3.3 Illustration of similar trajectories. One is from IC_j to MC_h and the other is IC_m to MC_h.

The following procedure is used to group trajectories.

Grouping Procedure:

Step 1: Construct the set of trajectories, $\underline{T} = \{T_1, T_2, ..., T_m\}$ based on the **Trajectory Determining Procedure** described above. Initialize a set of groups $\underline{G} = \{\varnothing\}$. Define a threshold value β_T. Set $i = 1$ and $k = 1$.

Step 2: If $i = 1$, assign the features of T_i as the feature of G_i. Otherwise set $i = i+1$.

Step 3: Compute all β_j between T_i and G_j ($j = 1, ..., k$), and find G_j which has the minimum β_j of all groups G_j ($j = 1, ..., k$). Let us assume the found G_j be G_l (i.e j = l). Its corresponding β_j is β_l.

Step 4: If $\beta_l \leq \beta_T$, the trajectory T_i is assigned to that group G_l. Update features of G_l with the features of T_i. Go to step 6.

Step 5: If $\beta_l > \beta_T$, T_i becomes the first member of a new group. Increase $k = k+1$. Assign the feature of T_i as the feature of G_k.

Step 6: Go to Step 2. This process continues until $i = m$.

A main purpose of grouping is to obtain simplified state and control information compared to cell data which contain a huge amount of information. Based on the simplified information, it is possible to generate fuzzy rules and membership functions. Cell states can be the IF-Part of a fuzzy rule and control levels can be the THEN-Part of a fuzzy rule.

There are multiple ways of using cell states and control levels of a group. One is to use a representative trajectory of a group, which may be the first trajectory of group,

the longest trajectory of a group, or others. Then, the cell states and control levels of the representative trajectory can be the IF-Part and THEN-Part of a fuzzy rule, respectively. The other way is to consider all cells of a group and extract their statistical properties. The statistical properties are the mean location and its standard deviation of cells in a group. Average control level of a group is also a statistical property of the group. Based on these statistical properties, we can create fuzzy membership functions corresponding to the various groups. We will discuss the construction of the fuzzy rules and membership functions in the next section.

3.4 Fuzzy Rule Base and Membership Functions based on Groups

Fuzzy rules and the associated membership functions are generated based on the statistical properties of groups. Figure 3.4 illustrates the generation of fuzzy membership functions which are projected onto input axes.

3.4.1 Fuzzy Rule Base

If we have k groups, $\mathbf{G}_j$ (j = 1,.., k), then the number of fuzzy rules is k. The generated rule base has the same structure as that shown in Figure 2.3 (b).

3.4.2 Membership Functions based on Groups

This Section explains how to construct the membership functions according to the description of Section 2.2.2. The shape of a membership function is either a symmetric trapezoidal function or a symmetric triangular function. To form these membership functions, the location of the middle point and the width of a triangle or trapezoid are needed.

The projections of the group onto the input axes become the fuzzy membership functions of IF-Part. The mean location is obtained by calculating the coordinates of all cells which are members of a group G_j. The mean location becomes the middle of the membership function. The standard deviation of the mean location a group is the width of the fuzzy membership function. Based on these two statistical values, a set of fuzzy premises can be created for a fuzzy rule base as in Equations (2.8) or (2.9).

The control level of a group represents the THEN-Part of a fuzzy rule. The membership functions of the THEN-Part of a fuzzy rule can be constructed by using the highest control level of a group or a weighted average of the two highest control levels of a group.

3.4.3 Fuzzy Controller

After designing the fuzzy rule base and the membership functions based on the groups of cells, a global fuzzy controller can be constructed. Constructing the global fuzzy controller consists of four main procedures described in Sections 2.2.4 through 2.2.7, which are fuzzification, implication, defuzzification, and aggregation. In the simulations described in Chapter 4, we use the following in constructing the fuzzy system: 1) The rule base is the Mamdani type, which is represented by Eq. (2.8), 2) The implication is the Min-operation in Eq. (2.11), 3) The defuzzification is based on the center of gravity, 4) The aggregation is the correlation minimum inference method represented by Eq. (2.13).

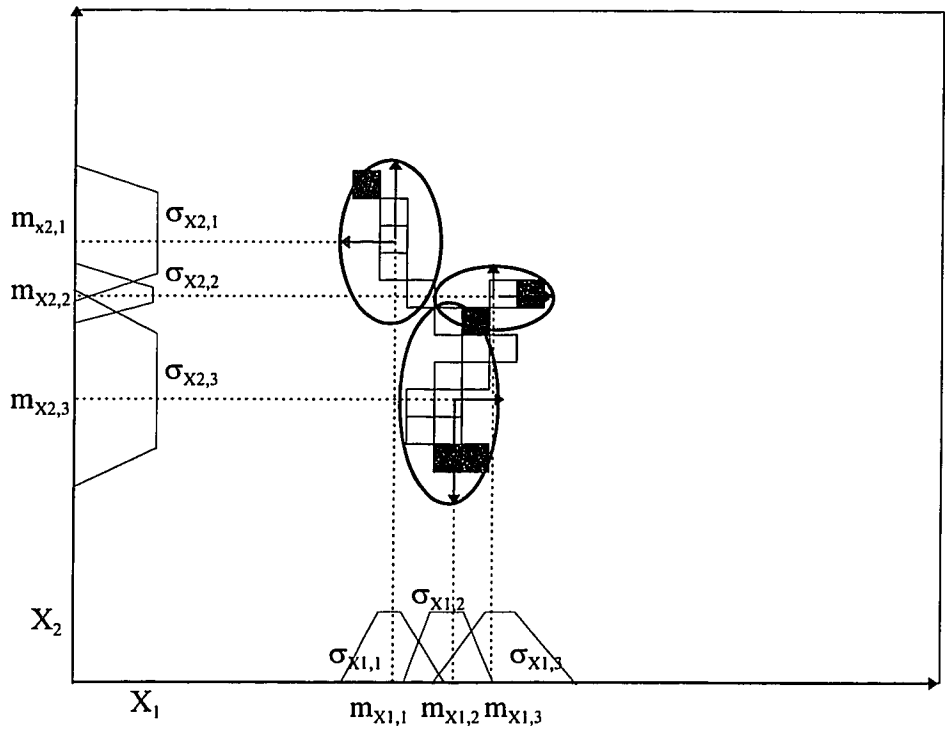


Figure 3.4 Each group defines a fuzzy rule region and its membership functions on the input axes.

CHAPTER 4

APPLICATION TO SHIP NAVIGATION AND CAR PARKING

In Chapter 3 we discussed the grouping of cells to generate fuzzy rules and membership functions of a fuzzy controller. In this Chapter we apply the proposed method to the ship navigation and car parking control problems. Sections 4.1 and 4.2 describe the ship navigation problem and the car parking control problem, respectively. It generates numerical information using cell-to-cell mapping. In this study, the cell order is rearranged to reduce the computational time and memory storage during the cell mapping procedure. Sections 4.3 and 4.4 show the grouping of cells and the generation of membership functions of a fuzzy controller. Simulation is performed to show the behavior of the fuzzy controller. The results are compared to those from the table-based controller.

4.1 Application to the Ship Navigation Problem

The ship navigation is first used in the numerical study. A classical example of this is Zermelo's problem [2, 3], in which the minimum path of a ship through an area of position-dependent current is searched for. The equations of motion are

$$\begin{aligned} \dot{x}_1 &= V\cos\theta + u(x_1, x_2) \\ \dot{x}_2 &= V\sin\theta + v(x_1, x_2) \end{aligned} \qquad (4.1)$$

where (x_1, x_2) are rectangular coordinates representing the position of the ship, u and v are the velocity components of the current in the x and y directions, respectively, V is the

ship speed relative to the water, which is a constant, and θ is the heading angle of the ship's axis relative to a fixed coordinate axis. In this example, it is assumed that u depends linearly on x_2 , v equals 0, and V equals 1. Figure 4.1 illustrates the ship navigation problem.

4.1.1 Optimal Cell Mapping

To generate optimal trajectories for this problem, first a cell structure has to be determined. The region of interest in the state space is taken to be the range [-1.05, 1.05] × [-1.05, 1.05] of the two state variables, X_1 and X_2. The specified region is partitioned into 21 by 21 cells with cell size 0.1 × 0.1. Thus the cell state space contains 441 regular cells (N_c = 441) and one sink cell which covers the entire area outside the region of interest. Sixteen discretized control levels are used: $u = j\pi/8$, j=0, 1, 2,..., 15. The sampling time t_m is chosen to be 0.01. The number of elements in the set of allowable time intervals is not prescribed but left open. A minimum time as described by equation (2.6) is used as a performance function (J_c) of the optimal strategy. The target cell is located at the center of cell state space in Figure 4.1.

4.1.2 Rearrangement of Cell Order around the Target Set

To speed up the cell mapping process, we order the cell sequence [4] as depicted in Figure 4.2. By this scheme of labeling, the cells representing the target and its neighbor region are not necessarily low in the cell numbers sequences. A low-number cell might take many mapping steps to arrive at the target. Since the sorting procedure starts from the target, many intermediate results may have to be stored during the computation until

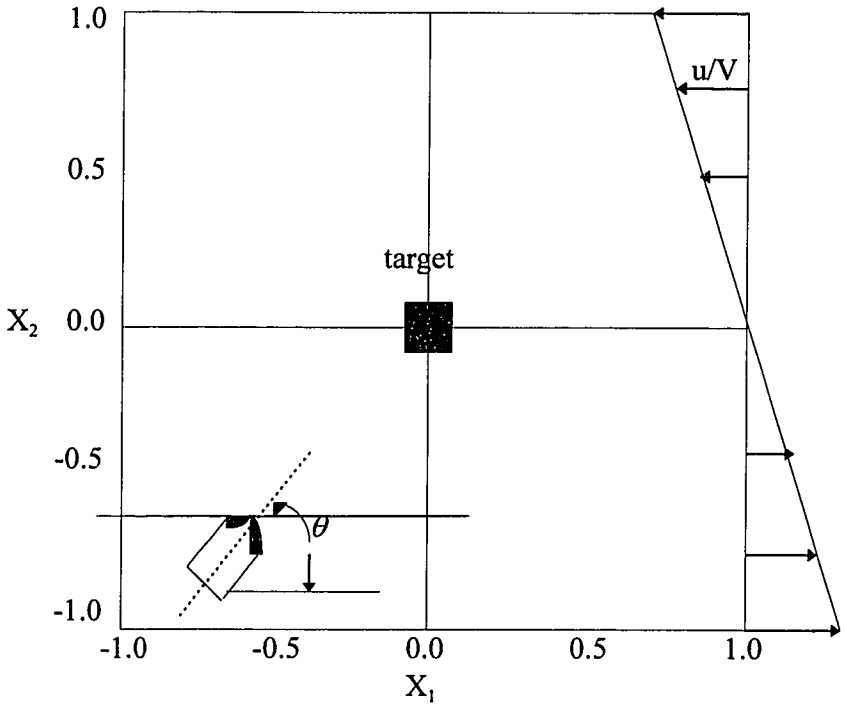


Figure 4.1 Ship navigation problem

the optimal routes are determined. We could lessen this computational burden by rearranging the order of the cells around the target cell.

As shown in Fig. 4.2, we apply the first wrapping layer which is a set of 2-dimensional rectangular cells immediately surrounding the target. Assume $X_1 = 10$ and $X_2 = 10$ for the target, then the cells of the first layer have $X_1 = 9$ or $X_1 = 11$ and $X_2 = 9$ or $X_2 = 11$. The second wrapping layer is a set of rectangular cells of the second layer cells have $X_1 = 8$ or $X_1 = 12$ and $X_2 = 8$ or $X_2 = 12$. The higher number layers are defined in a similar way.

In the wrapping layers, we arrange the cells by following sequence: First we number cells in the negative X_1 direction, followed by the positive X_1 direction with respect to the target cell. Second, cells are numbered from the negative X_2 direction to the positive X_2 direction. Proper care should be taken, of course, to include the corner cells of the layer such as cell numbers 13, 14, 15, 16 in the wrapping layer 2.

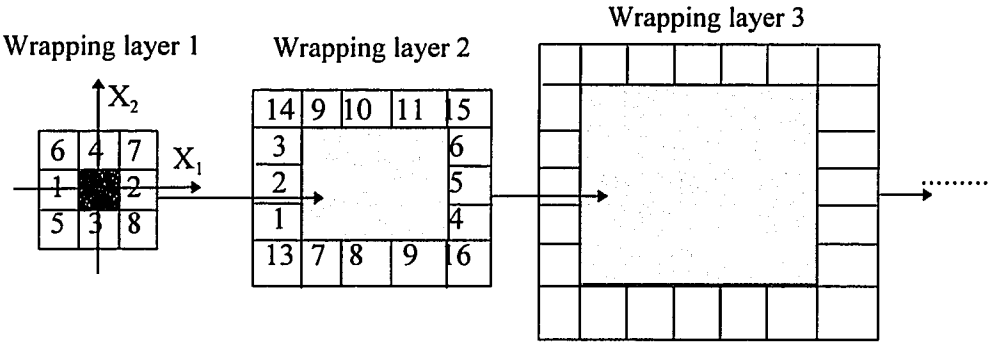


Figure 4.2 Arrangement of the cell processing order around the target with wrapping layers

4.1.3 Optimal Cell Mapping Table

After optimal cell mappings, all the 441 regular cells (represented by rectangles in Figure 4.3) are found to be controllable cells. Table 4.1 shows the selected cell data from the obtained control table. The fields of the Table 4.1 are referred to Section 2.1.3. The coordinates of each cell are represented by (X_1, X_2). The fields of the target cell, z_0, are located at the first row of the table. Cell $\mathbf{z}_0$ is located at (10, 10) of the cell coordinates and its corresponding real coordinates are located at (0.0, 0.0). The control of the target cell is 0 radian which comes from $u = j\pi/8$ with $j = 0$. We set the step number, s_i, and the image cell number, I_i, of the target cell as 0, respectively.

Cell $\mathbf{z}_9$ is located at (12, 8) of the cell coordinates. It maps to the image cell z_1 with the control of 2.355 rad which comes from $u = j\pi/8$ with $j = 6$. Two steps are needed for it to move to the target. Cell $\mathbf{z}_{206}$ maps to the image cell z_{152} with the control of 5.495 rad steering angle. It needs 7 steps to reach the target.

Figure 4.3 shows the resulting optimal trajectories obtained by the cell-to-cell mapping method. The circles in the figure are the center points of the discretized rectangular cells. A line between two neighboring circles represents the connection of a domain cell to its image cell. Figure 4.4 shows the optimal control angle of each cell corresponding to Figure 4.3. Each cell may have one of the sixteen discretized control levels.

Table 4.1 Optimal control cell data for the ship navigation problem

z_i	$X_{1,i}$	$X_{2,i}$	u	s_i	I_i
0	10	10	0	0	0
1	11	9	6	1	0
2	9	9	2	1	0
3	10	9	3	1	0
4	11	10	7	1	0
5	11	11	10	1	0
6	9	11	14	1	0
7	10	11	11	1	0
8	9	10	0	1	0
9	12	8	6	2	1
10	8	8	2	2	2
11	9	8	3	2	3
12	10	8	4	2	3
13	11	8	6	2	3
14	12	9	7	2	1
15	12	10	7	2	4
16	12	11	8	2	5
17	12	12	10	2	5
18	11	12	11	2	7
		.....	.....	.....	
206	8	17	14	7	152
207	7	17	14	7	153
208	6	17	14	7	154
		.....	.....	.....	
427	6	0	5	10	347
428	5	0	4	10	348
429	4	0	4	10	349
430	3	0	4	10	350
431	2	0	4	10	351
432	1	0	4	10	352
433	0	0	4	10	353
		.....	.....	.....	

z_i: Cell order, I_i is image cell number, u is discrete control level, s_i is step numbers to the target

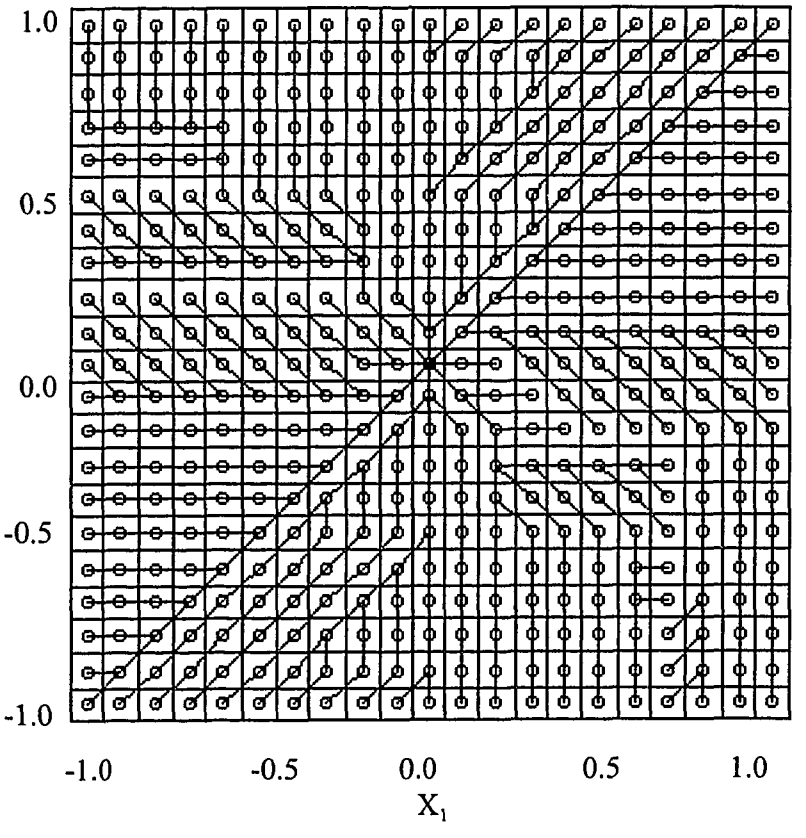


Figure 4.3 Optimal trajectories generated by cell mapping method.

11	14	14	14	14	14	14	14	14	14	14	13	13	13	13	12	12	12	12	12	12
11	13	13	13	14	14	14	14	14	14	13	13	13	13	12	12	12	12	12	12	11
10	13	13	13	14	14	14	14	14	14	13	13	13	12	12	12	12	12	12	11	11
0	0	0	0	14	14	14	14	14	14	13	13	12	12	12	12	12	12	11	11	11
0	0	0	0	14	14	14	14	14	14	13	12	12	12	12	11	11	10	10	10	10
15	15	15	15	15	15	15	15	14	14	12	12	12	12	11	11	10	10	10	10	10
15	15	15	15	15	15	15	15	14	14	12	12	12	11	11	10	10	10	10	10	10
15	15	15	15	15	15	15	15	14	14	12	12	11	10	9	9	9	9	9	9	9
14	14	14	14	14	14	14	14	14	14	12	11	10	9	9	9	9	9	9	9	9
14	14	14	14	14	14	14	14	14	14	11	10	8	8	8	8	8	8	8	8	8
14	14	14	14	14	14	14	14	0	0	0	7	7	6	6	6	6	6	6	6	6
0	0	0	0	0	0	0	0	0	2	3	6	7	7	6	6	6	6	6	6	6
1	1	1	1	1	1	1	1	2	3	4	6	6	7	7	6	6	6	6	6	6
1	1	1	1	1	1	1	2	3	4	4	6	6	7	7	7	7	7	6	6	6
2	2	2	2	2	2	3	3	4	4	4	6	6	7	7	7	7	7	6	6	6
2	2	2	2	2	3	3	4	4	4	4	6	6	7	7	7	7	7	6	6	6
2	2	2	2	3	3	4	4	4	4	5	6	6	6	6	6	6	8	6	6	6
3	3	3	4	4	4	4	4	4	5	5	6	6	6	6	6	6	8	6	6	6
3	3	4	4	4	4	4	4	5	5	5	6	6	6	6	6	6	4	6	6	6
3	4	4	4	4	4	4	5	5	5	5	6	6	6	6	6	6	4	6	6	6
4	4	4	4	4	4	5	5	5	5	6	6	6	6	6	6	6	5	6	6	6

Figure 4.4 Optimal control angles of cells in the region of $-1.05 < X_1 < 1.05$ and $-1.05 < X_2 < 1.05$.

4.2 Application to the Car Parking Control Problem

In this section, a car parking control problem is considered. Figure 4.5 shows the simulated car and its loading dock. The state variables representing the vehicle are x, y, and ϕ, which are the Cartesian coordinates of the center of the rear wheels and the angle of the vehicle with respect to the horizontal x axis. The objective of this control problem is to move the center of the rear wheels having coordinates (x, y) as close as possible to the center of the loading dock having coordinates (x_d, y_d), with the vehicle perpendicular to the dock, i.e., $\phi_d=90^0$, at the target position.

The main difference in the problem formulation between the previous studies of the car parking control problem and the study described here is as follows: The previous studies allowed only backward movements of the vehicle while our study also allows forward movements. Because only backward movements were allowed of the vehicle, the previous researchers assumed enough clearance between the vehicle and the loading dock, and thus the final y coordinate of the vehicles was ignored. Thus, their controller had two inputs ϕ and x and produced the steering angle θ. The rules of their fuzzy controllers were generated by human experts using the vehicle trajectories obtained experimentally with the vehicle at various initial positions under different steering angles. Kosko and Gong [13] applied FAM (Fuzzy Associate Memory) to generate a fuzzy controller. The extracted trajectories were decoded into linguistic knowledge to design the fuzzy controller. Lin and Lee [15] developed a fuzzy-neural network controller and applied it to the car control problem.

In this dissertation study, a cell based planner is applied to replace human experts. It generates all the possible optimal trajectories which cannot be achieved by human experts in a systematic way. Also, instead of neglecting the y variable, one forward movement of the vehicle is allowed to enable the vehicle to reach the target from large ranges of initial position and orientation of the loading zone.

4.2.1 Discrete Cell Space

To generate optimal trajectories, a cell structure is again determined first. The region of interest in the state space is taken to be the ranges of state variables of x, y, and ϕ, which are $[x_{min}, x_{max}]$, $[y_{min}, y_{max}]$, and [-90, 270], respectively. The cell state space $\boldsymbol{Z}$ is the closed rectangloid:

$$\boldsymbol{Z} = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [-90, 270]$$

x_{min} and y_{min} are both set to 0, x_{max} as 12, and y_{max} as 6, and thus the cell state space is $[0, 12]\times[0, 6]\times[-90, 270]$. From Eq. (2.2) the dimensions of each cell are as follows: $\Delta x = x_2 - x_1 = 12/27$, $\Delta y = y_2 - y_1 = 6/14$, and $\Delta\phi = \phi_2 - \phi_1 = 2\pi/27$. The total number of cells in the cell state space is 10,206 ($N_c = 27 * 14 * 27$).

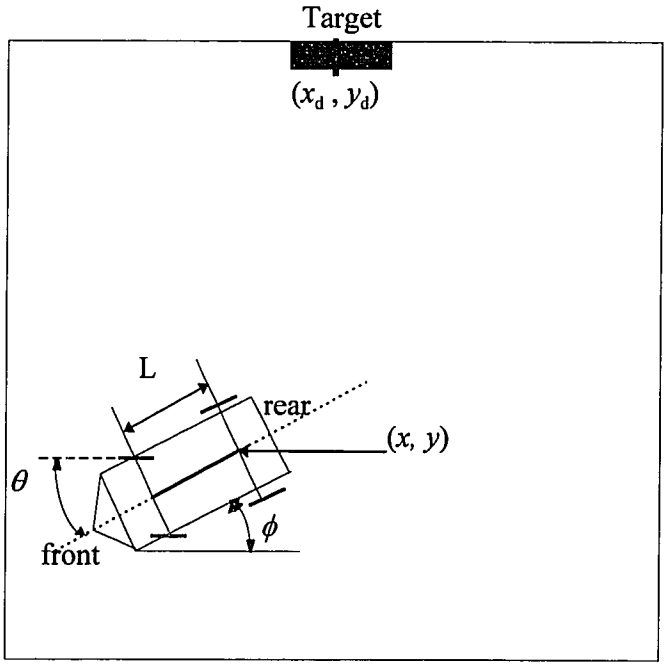


Figure 4.5 The vehicle and its loading zone.

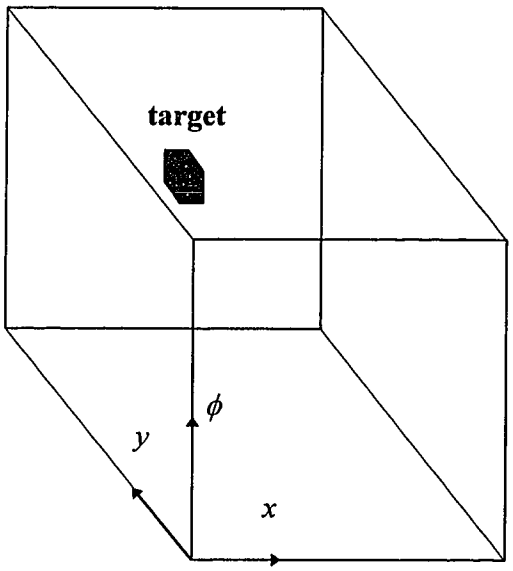


Figure 4.6 3-D cell state space for the vehicle parking

4.2.2 Target Set of Cells

After the discrete cell state space is constructed, the goal (or target) cell states must be given. The cell coordinates of the target cell are [13, 13, 13], which represent the loading dock of the vehicle. Figure 4.6 shows the 3-D cell state space for the vehicle parking and target cells.

4.2.3 Kinematics of Car-Like Vehicle and Controls

Let us parameterize the vehicle path by t, the elapsed time from the beginning of the motion. Given a steering angle θ and a velocity v of the rear center point in the Figure 4.5, the velocity parameters of the center of the vehicle are:

$$\dot{x} = v\cos\phi,\ \dot{y} = v\sin\phi,\ \dot{\phi} = \frac{v}{L}\tan\theta. \qquad (4.2)$$

where L is the distance between the front wheel and the center of the rear wheels. Integration of these equations leads to

$$\phi(t) = \phi(0) + \frac{v}{L}(\tan\theta)t,$$

$$x(t) = x(0) + \frac{L}{\tan\theta}(\sin(\phi(0) + \frac{vt}{L}\tan\theta) - \sin(\phi(0))), \qquad (4.3)$$

$$y(t) = y(0) - \frac{L}{\tan\theta}(\cos(\phi(0) + \frac{vt}{L}\tan\theta) - \cos(\phi(0))),$$

The finite dimensional vector U, which is the set of discrete steering angles is given by [-30^0, -15^0, 0^0, 15^0, 30^0] and its unit is degree. We also assume L to be 0.8 unit, and the moving distance at each sampling time, Δt, to be 1/4 of L.

4.2.4 Rearrangement of Cell Order around the Target Set

To speed up the cell mapping process, we order the cell sequence [4] as depicted in Figure 4.7. By this scheme of labeling, the cells representing the target and its neighbor region are not necessarily low in the cell numbers sequences. A low-number cell might take many mapping steps to arrive at the target. Since the sorting procedure starts from the target, many intermediate results may have to be stored during the computation until the optimal routes are determined. We could lessen this computational burden by rearranging the order of the cells around the target cell.

As shown in Fig. 4.7, we apply the first wrapping layer which is a set of 3-dimensional rectangular cells immediately surrounding the target. Assume y = 13 for the target, then the cells of the first layer have y =13 or y=12. The second wrapping layer is a set of rectangular cells and the second layer cells have y = 13, 12 or 11. The higher number layers are defined in a similar way.

In the wrapping layers, we arrange the cells by the following sequence: First we number cells in the negative x direction, followed by the positive x direction with respect to the target cell. Second, cells are numbered from the negative ϕ direction to the positive ϕ direction. Third, cells displaced in the y direction are numbered. Proper care should be taken, of course, to include the corner cells of the layer such as cell numbers 13, 14, 15, 16 in the wrapping layer 2.

4.2.5 Discriminate Rule (or sorting) based on the Objective Function

During the sorting procedure, which is equation (2.6) in Section 2.1, we use minimization of the path length as the object function. If there are two shortest paths to the target, then the one having a smaller number of control switching is selected.

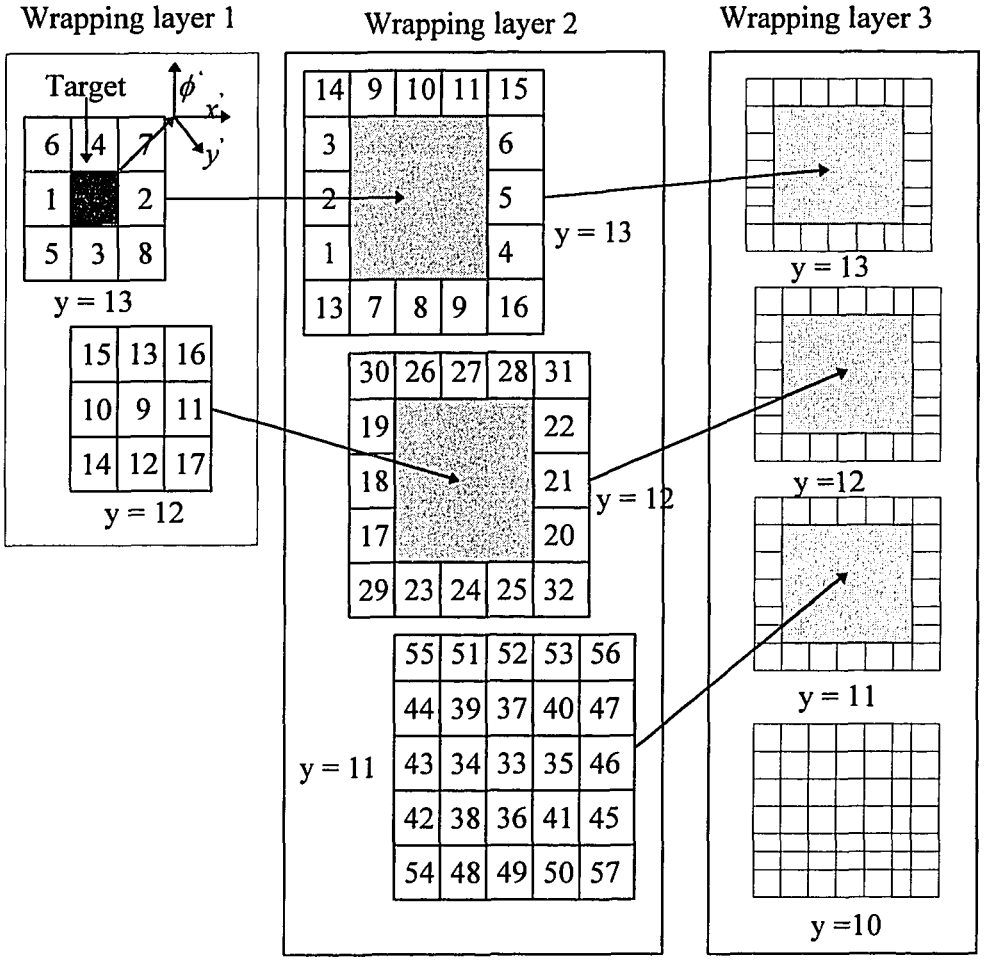


Figure 4.7 Arrangement of the cell processing order around the target with wrapping layers

4.2.6 Selection of Moving Direction of Vehicle

One of our goals of cell mapping is to get global control information such that the optimal paths are not very sensitive to the initial state. For this purpose, first only a backward movement of the vehicle is allowed until all the cells are processed. Among the processed cells, one set of cells is determined as reachable cells in their paths are connected to the target cells. The other set of cells are unreachable cells (i.e. cells which cannot reach the target). The paths of the unreachable cells are not connected to the target but to the sink cell (outside the domain region) because of the dynamic and other physical constraints, i.e., limitation of steering angle of the vehicle and the closed walls of the loading zone. If there are unreachable cells after the process allowing backward movements, then we allow forward movement of a vehicle to the previously obtained reachable cells.

4.2.7 Cell Mapping Results

After cell mapping of the vehicle parking problem, a total of 5,229 cells are found as controllable cells to the target cell when one switching of the vehicle moving direction is allowed. Among them, cells numbered 0 through 2,371 are determined as reachable cells to the target with only backward motion.

Cells numbered 2,372 to 5,229 need initially forward motion and then backward motion to reach the target. They are determined as follows: initially moving forward until they are connected to the previously determined cells which can reach the target with only backward motion, i.e. cells 0 through 2,371. The quantity of information obtained

in this problem is so large that it is impossible to express the result like Figures 4.3 and 4.4 in the ship navigation problem.

Table 4.2 shows the selected cell data from the obtained control table. The fields of the Table 4.2 are referred to in Section 2.1.3. The field of ***movedir*** is a special field for this particular problem. It represents the moving direction of a car. Backward has 0 value and forward has 1, respectively. The coordinates of each cell are represented by (x_i, y_i, ϕ_i). For example, cell $\mathbf{z}_8$ is located at (13, 13, 5) of the cell coordinates. It maps to the image cell z_7 with the control of 0 degree and only backward movement. Eight steps are needed for it to move to the target which is z_0. Cell $\mathbf{z}_{2,371}$ maps to the image cell $z_{2,345}$ with the control of 30 degree steering angle and only backward motion. It needs 17 steps to reach the target. The trajectory from cell $\mathbf{z}_{2,373}$ moves to the image cell z_1 with 30 degree steering angle and forward moving direction. Then, it moves to the target with the control of 0 degree and backward moving direction. We can visualize the above mentioned mapping with Figures 4.8 and 4.9. Figures 4.8 through 4.9 show the results with 10 different initial vehicle positions (x, y, ϕ).

Table 4.2 Optimal control cell data for the car parking problem

$z_i(o_i)$	x	ϕ	y	u	s_i	I_i	*movedir*
0	13	13	13	0	0	0	0
1	13	13	12	0	1	0	0
2	13	13	11	0	2	1	0
3	13	13	10	0	3	2	0
4	13	13	9	0	4	3	0
5	13	13	8	0	5	4	0
6	13	13	7	0	6	5	0
7	13	13	6	0	7	6	0
8	13	13	5	0	8	7	0
9	13	13	4	0	9	8	0
10	13	13	3	0	10	9	0
11	13	13	2	0	11	10	0
12	13	13	1	0	12	11	0
13	13	13	0	0	13	12	0
14	13	12	12	-30	1	0	0
15	13	14	12	30	1	0	0
			.....	.....			
2,365	26	16	0	-15	15	1719	0
2,366	26	17	0	-15	15	1720	0
2,367	26	18	0	30	15	2341	0
2,368	26	19	0	30	15	2342	0
2,369	26	20	0	30	15	2343	0
2,370	26	21	0	30	16	2344	0
2,371	26	22	0	30	17	2345	0
2,372	13	12	13	30	1	1	1
2,373	13	14	13	-30	1	1	1
2,374	13	11	13	30	2	14	1
2,375	13	15	13	-30	2	15	1
			.....	.....			
5,225	22	26	0	-30	19	4350	1
5,226	25	26	0	-30	23	4353	1
5,227	0	0	0	30	19	4822	1
5,228	0	26	0	-30	19	4837	1
5,229	26	26	0	-30	23	4849	1

z_i : cell order, I_i is image cell number, u is discrete control, s_i is step numbers to the target

Figure 4.8 shows the car trajectories for 5 different initial locations, which are located at x = 1.6, y = 1.6, and ϕ = -80 + 85 * i (i =0..4). The corresponding cell coordinates of these locations are z_{xi} =4 and z_{yi} = 4. The purpose of using these locations is to set the vehicle close to the boundary of the loading dock. For example, at the position of Figure 4.8 (b), where x =1.6, y = 1.6, and ϕ = 5 degrees, the car successfully arrives at the desired target with only backward motion. At the position of Figure 4.8 (e), where x =1.6, y = 1.6, and ϕ = 260 degrees, the car initially moves forward 14 steps to avoid collision with the left wall of the loading dock and changes the moving direction to backward.

Figure 4.9 shows the car trajectories for the other 5 different initial locations, which are x =1.6, y = 4.0, and ϕ = -80 + 85*i (i =0..4). For example, at x =1.6, y = 4.0, and ϕ = -80 degrees, the car successfully arrives at the desired position with only backing motion. At the location shown in Figure 4.9 (c), x =1.6, y = 4.0, and ϕ = 90 degrees, the car initially moves forward instead of backward. The car then turns right near the left-lower corner of the loading zone. Then, it moves backward to the target. At the location of Figure 4.9 (e), x =1.6, y = 4.0, and ϕ = 260 degrees, the vehicle initially moves forward many steps to avoid collision with the upper wall of the loading dock and then changes the moving direction near the loading dock.

Figure 4.10 shows the reachable cells in the discrete cell state space. The cell regions of black colored represent reachable cells. The white colored regions are unreachable regions. Figure 4.11 represents the reachable cells with backward movements. The cell regions of black color represent backward movements of a car.

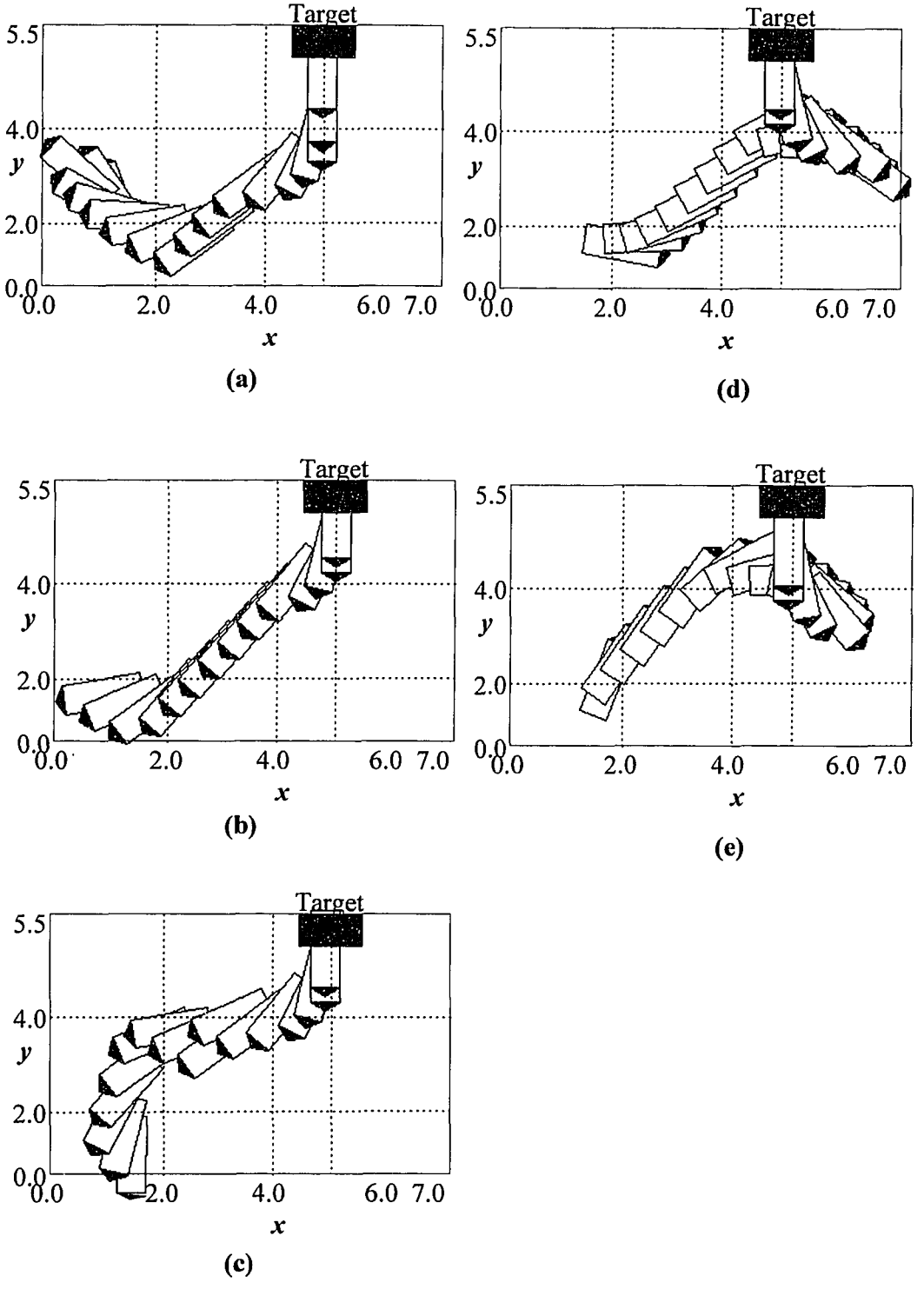


Figure 4.8 Car paths for 5 different initial locations. The car starts from the same position but different orientations.

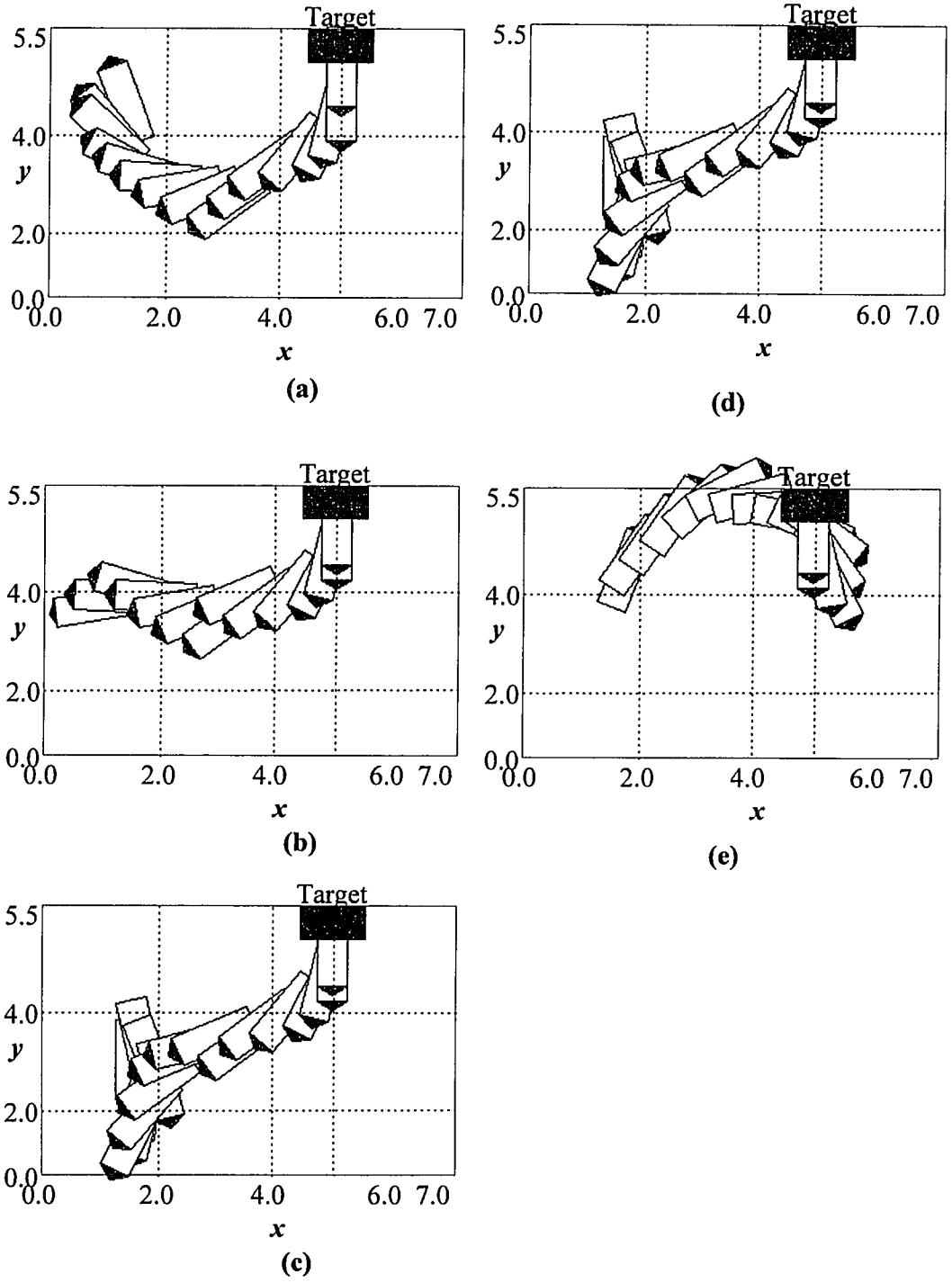


Figure 4.9 Car paths for 5 different initial locations. The car starts from the same position but different orientations.

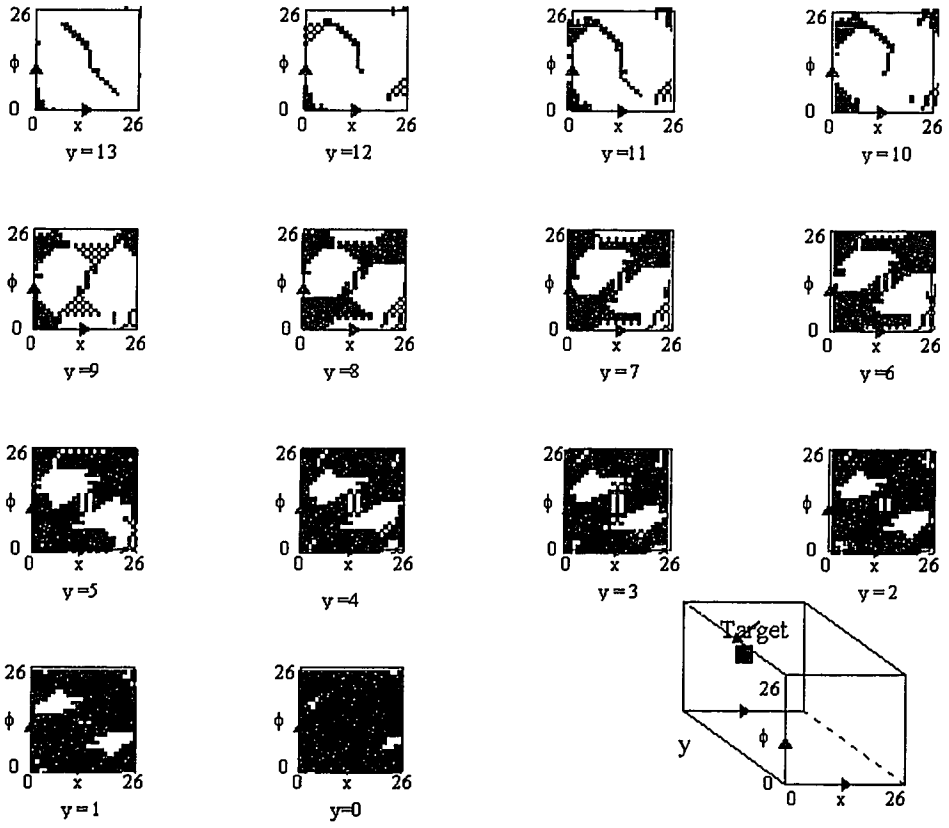


Figure 4.10 Reachable cells in the discrete cell state space. The cell regions of black color represent reachable cells. The white colored regions are unreachable regions.

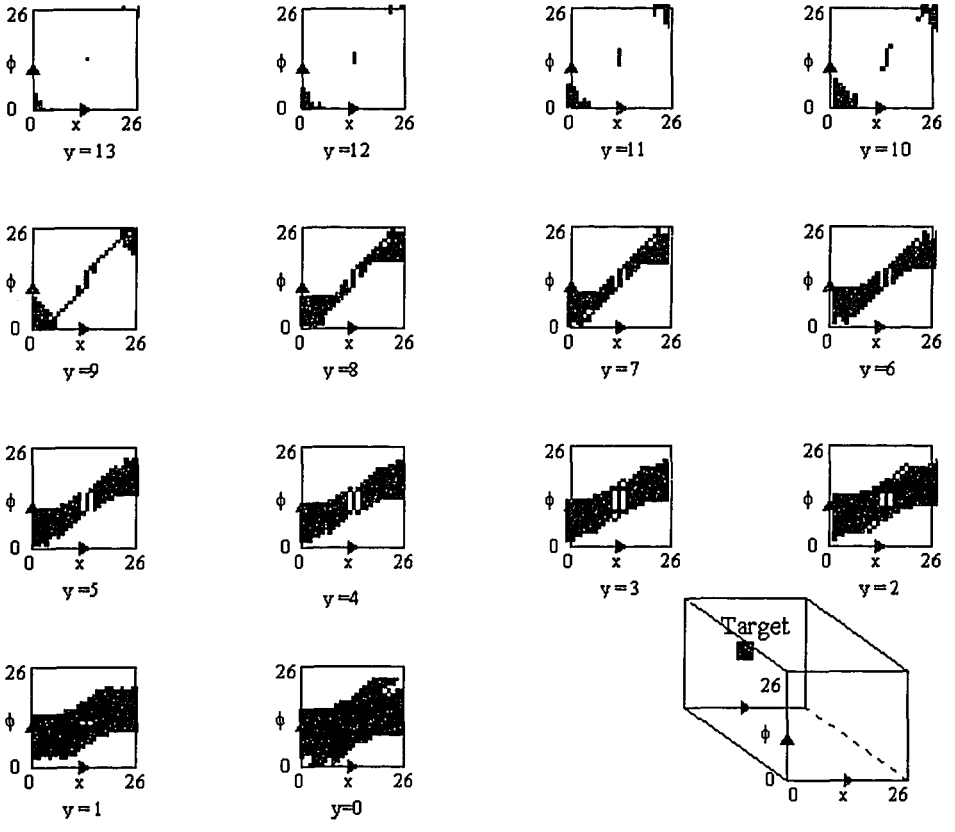


Figure 4.11 Reachable cells of backward movements. The cell regions of black color represent backward movements of a car.

4.3 Grouping Trajectories and the Generated Fuzzy Controller of the Ship Navigation Problem

This section shows the simulation results of the generated fuzzy controller for the ship navigation problem that is dealt with in Section 4.1. We will first discuss the construction of trajectory groups. Then, the performance of the generated fuzzy controller will be shown. The total number of cells is 441 as described in Section 4.1. The total number of trajectories is 108 using the **Trajectory Determining Procedure** described in Section 3.1. Among them, there are 91 trajectories which are type I and 17 trajectories that are type II. We use $\beta_T = 3.5$ as the threshold value. The generation of trajectory groups is performed with the **Grouping Procedure** described in Section 3.3. There are 51 groups of trajectories. The total number of cells contained in these groups of trajectories is 212. Figure 4.12 shows the resultant grouping of the optimal trajectories.

After designing the fuzzy rule base and the membership functions based on the groups, a global fuzzy controller is constructed. We use the hightest control level of each group in the THEN-Part of a fuzzy rule in Section 3.4.2. In the simulations, we choose the following for the fuzzy system: 1) The rule base is the Mamdani type , 2) The implication is the Min-operation, 3) The defuzzification is based on the center of gravity, 4) A symmetric trapezoidal function is used for the shape of the membership function of the IF-Part and a symmetric triangular function is used for the shape of the membership function of the THEN-Part. Figure 4.13 shows the rule base of the constructed fuzzy controller with the membership functions in X_1 , X_2 , and u axes based on groups.

Figure 4.14 shows the trajectories to the target from 16 different initial locations. It illustrates that the motion simulation results of using the fuzzy controller approximate

those of the table-based optimal controller in Section 4.1. In all of the 16 initial locations, which are represented by the black circles, the ship is able to successfully move to the target region which is represented by the black rectangle.

Figure 4.15 (a) represents the control activity of a selected trajectory starting from location #16. The horizontal axis is the number of moving steps of the ship from the initial location to the target. The vertical axis is the steering angles in the range between 0 and 2π. The control action is depicted by the steering angle between 3.92 radians and 4.31 radians for the trajectory, similar to that shown in Fig. 4.4.

Figure 4.15 (b) shows the number of rules used in each moving step. For example, 4 fuzzy rules are used at the fifth moving step of the ship.

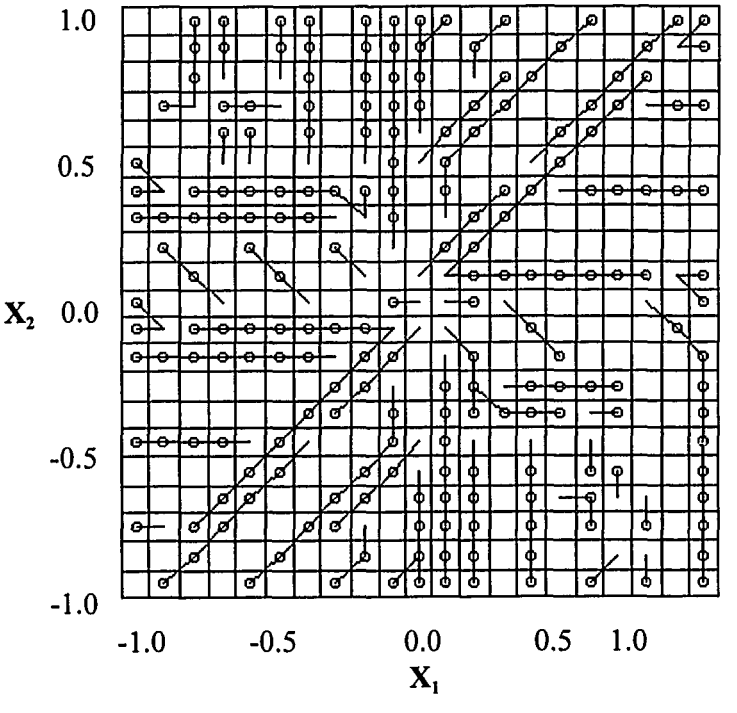


Figure 4.12 Selected trajectories and their cells by grouping trajectories

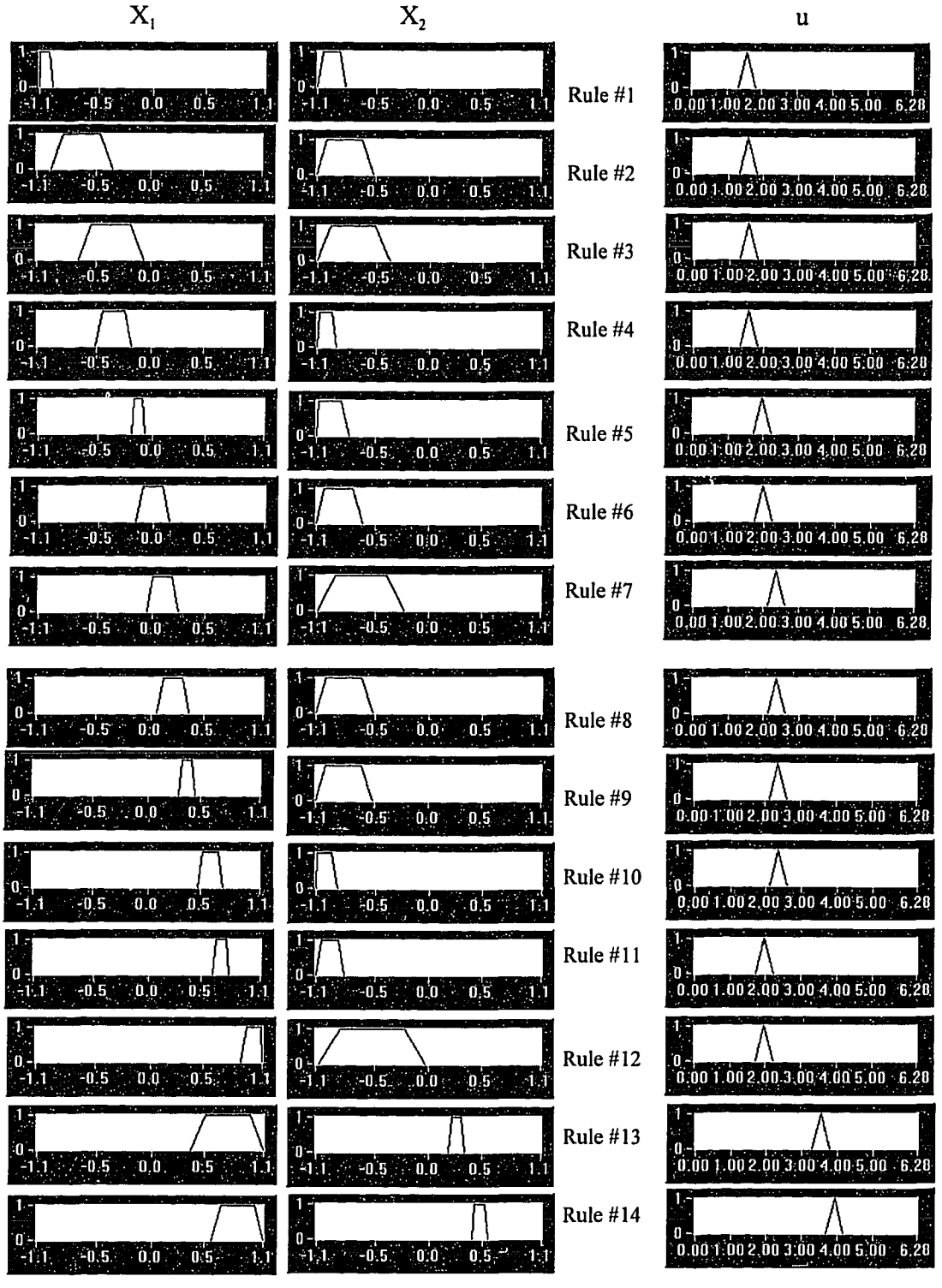


Figure 4.13 The rule base of a global fuzzy control

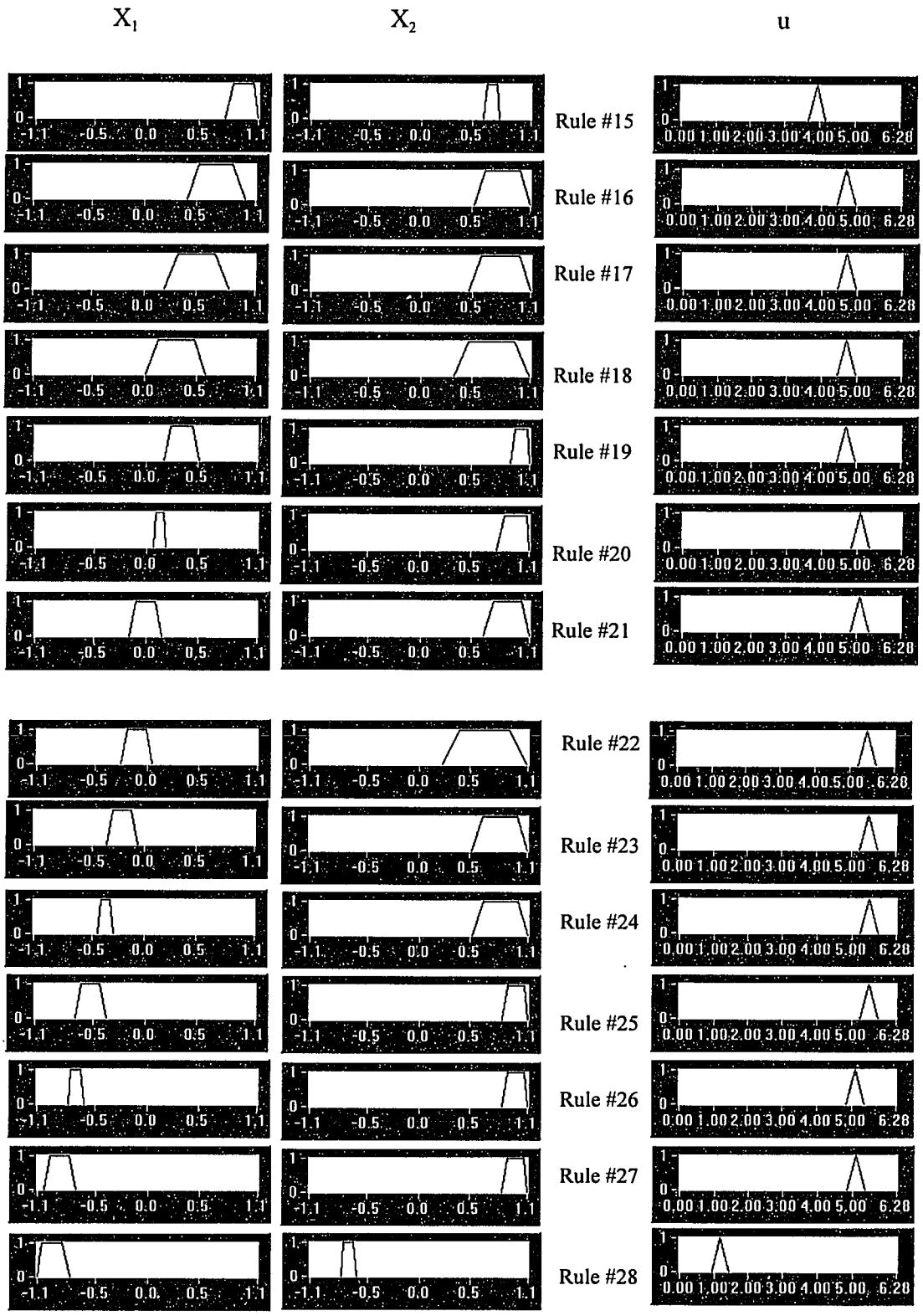


Figure 4.13 The rule base of a global fuzzy control (continued)

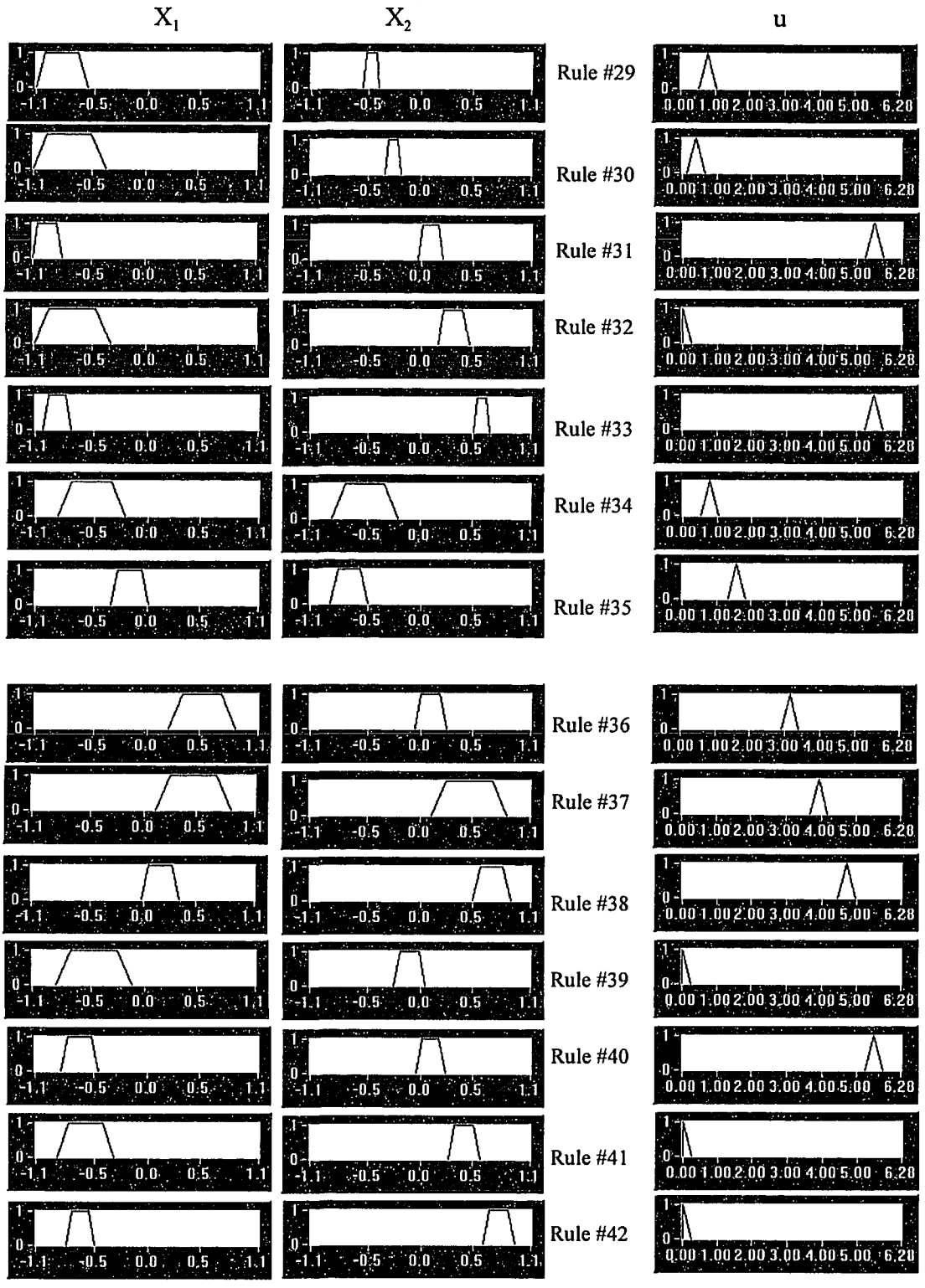


Figure 4.13 The rule base of a global fuzzy control (continued)

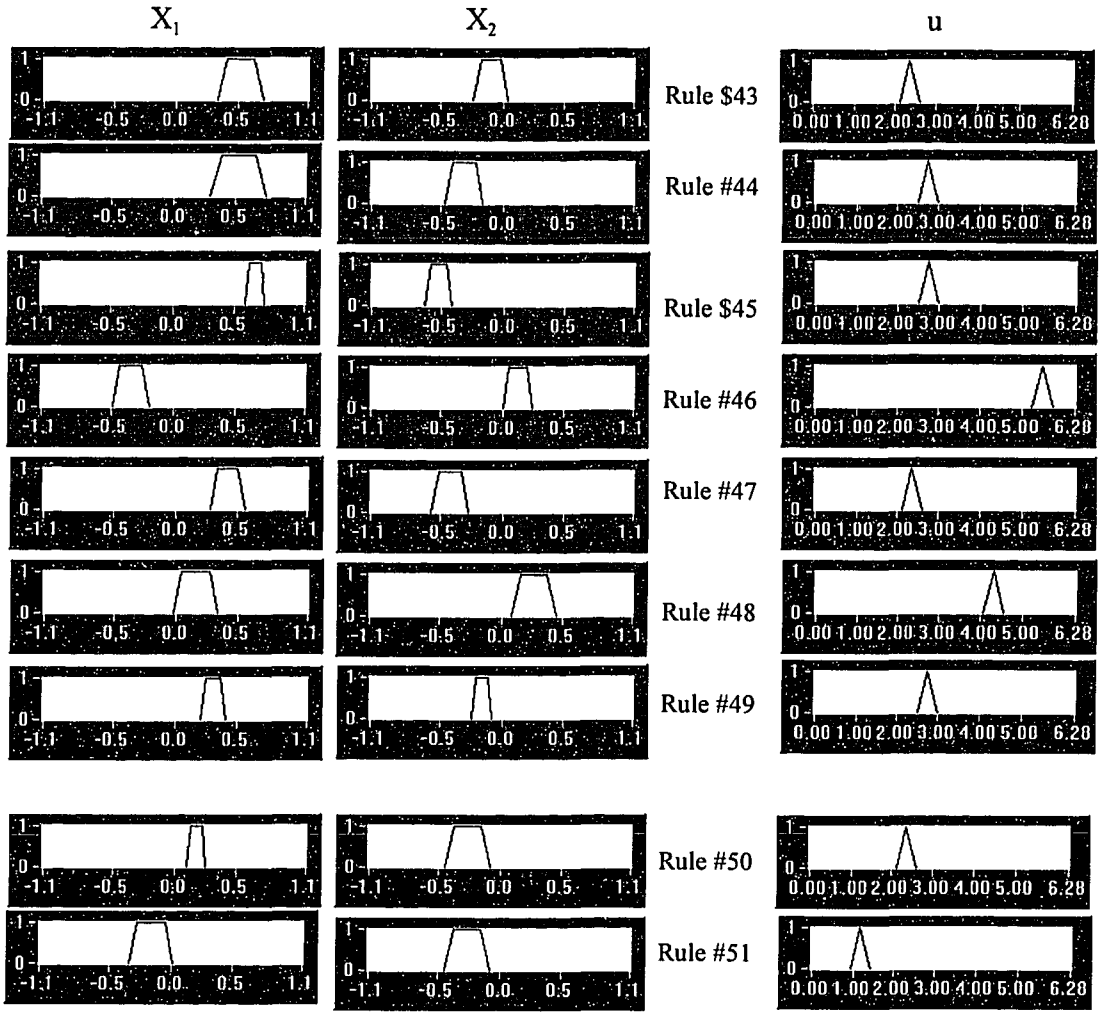


Figure 4.13 The rule base of a global fuzzy control (continued)

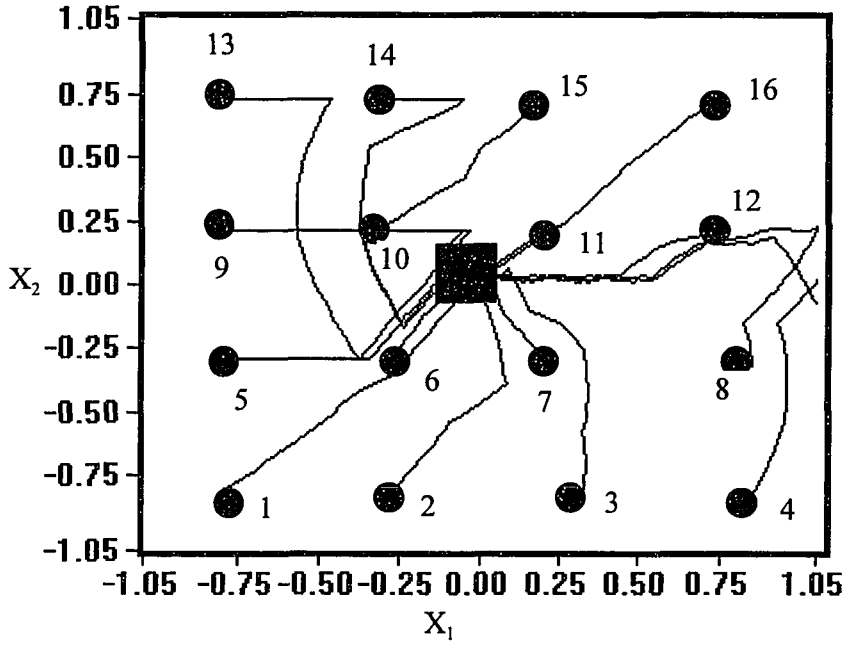
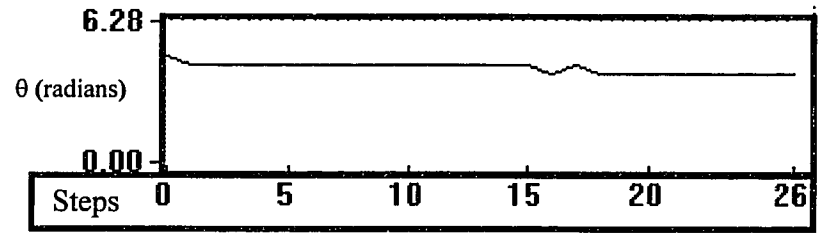
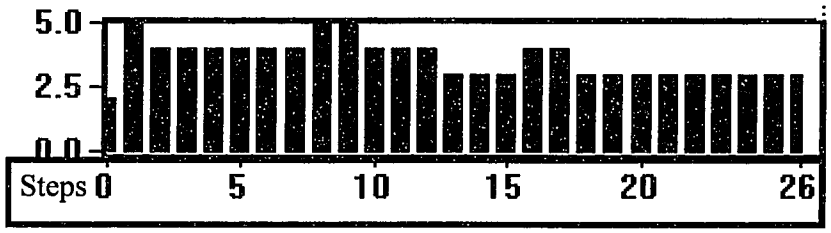


Figure 4.14 Simulated trajectories for the ship navigation control with 16 initial different locations by the global fuzzy controller.



(a)

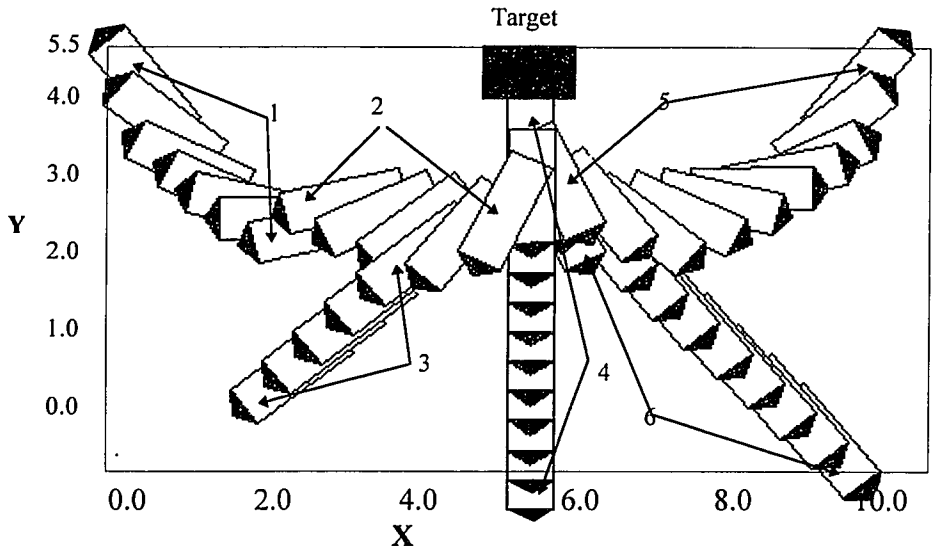


(b)

Figure 4.15 (a) Control activity of a trajectory starting from location #16 in Fig. 4.14. (b) The number of rules used in each step.

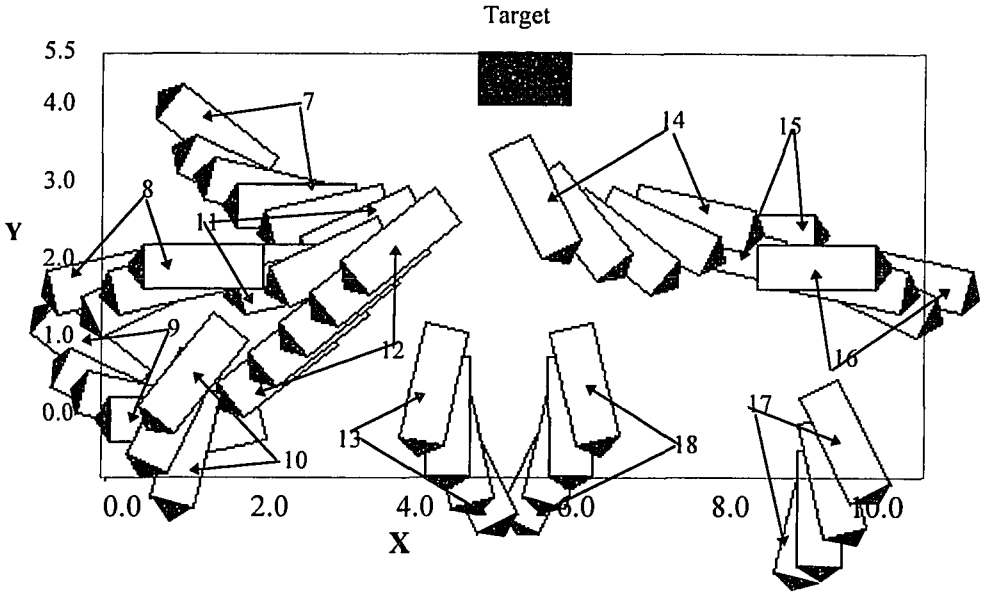
4.4 Grouping Trajectories and the Generated Fuzzy Controller of the Car Parking Control Problem

This section shows the simulation results of the car parking control problem which is dealt with in Section 4.2. Again, we will discuss that trajectory groups and the performance of the generated fuzzy controller. After grouping of the trajectories, there are 68 groups of trajectories. The threshold value β_T is set at 3.0. Among the 68 groups, 31 groups of trajectories are shown in Figure 4.16. Figure 4.17 shows the membership functions of the car parking fuzzy controller.

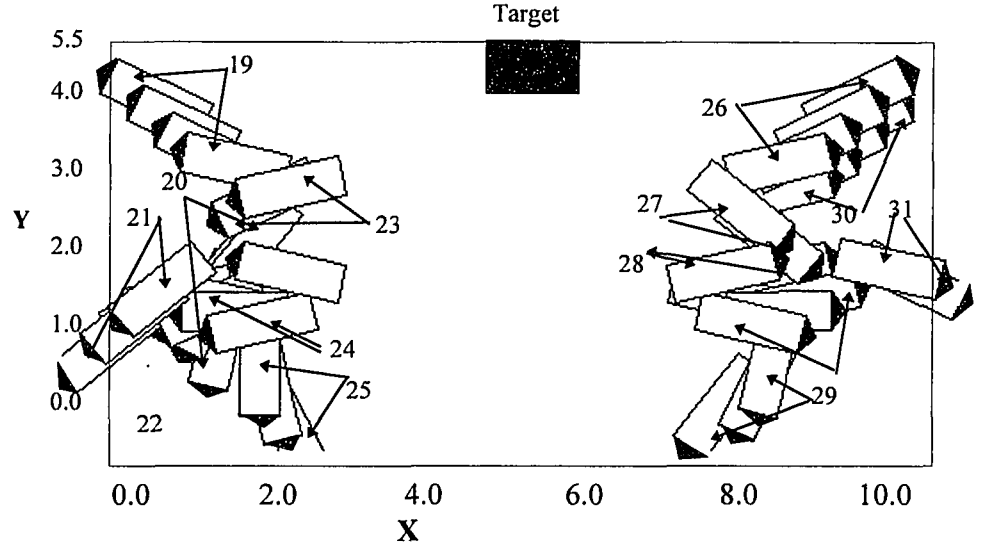


(a)

Figure 4.16 Representative groups of trajectories based on the cell mapping. (a) shows the 6 longest trajectories, (b) shows the next 12 longest trajectories, and (c) shows the next 13 longest trajectories.

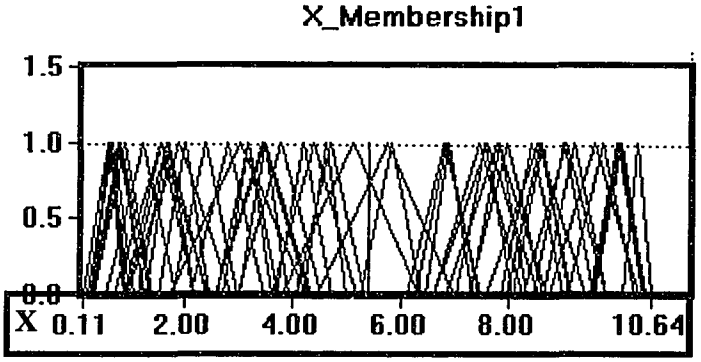


(b)

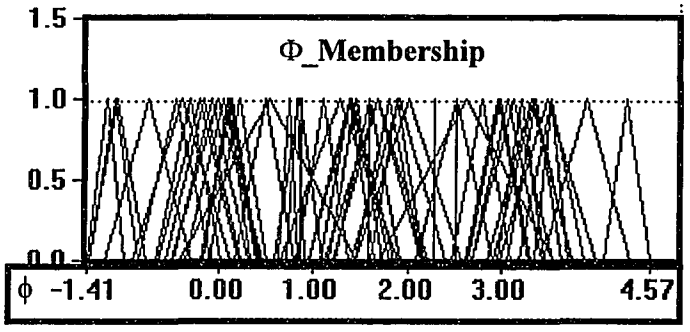


(c)

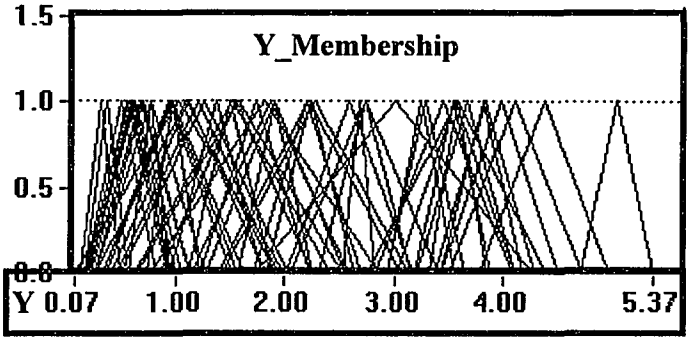
Figure 4.16 Representative groups of trajectories based on the cell mapping (continued). (a) shows the 6 longest trajectories, (b) shows the next 12 longest trajectories, and (c) shows the next 13 longest trajectories.



(a)



(b)



(c)

Figure 4.17 Membership functions of the car parking fuzzy controller. (a) x axis (b) ϕ axis. (c) y axis

Figures 4.18 and 4.19 compare the simulated trajectories using the table-based controller with those using the fuzzy controller for the car parking problem. There are 14 different trajectories starting from different locations in the parking lot. For most of the trajectories, the result of the optimal fuzzy controller approximates that of the table based optimal controller, with 68 rules used in the fuzzy controller and a data table of 2,371 cells in the table-based controller. The table-based controller fails to reach the target from locations 8, 11, and 12. The fuzzy controller fails to reach the target from locations 8, 11, 12 and 14.

Figure 4.20 represents the control activity of a sample trajectory which starts from location #2 of Figures 4.18 and 4.19. The horizontal axis is the number of moving steps of the car from the starting location to the target location. The vertical axis shows the steering angles. Both the steering angle and the number of steps to the target are similar for the two types of optimal controllers.

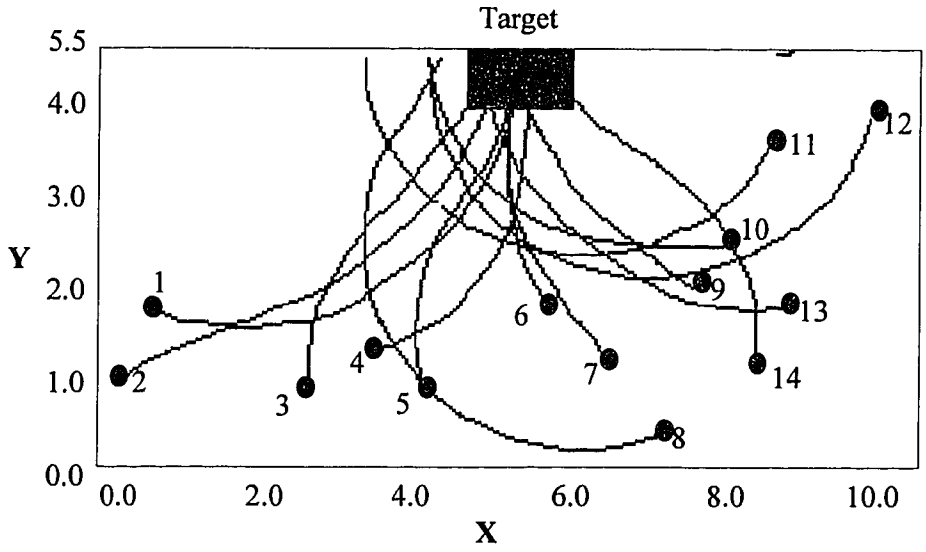


Figure 4.18 Simulated trajectories for the car parking control problem with 14 different locations using table-based controller.

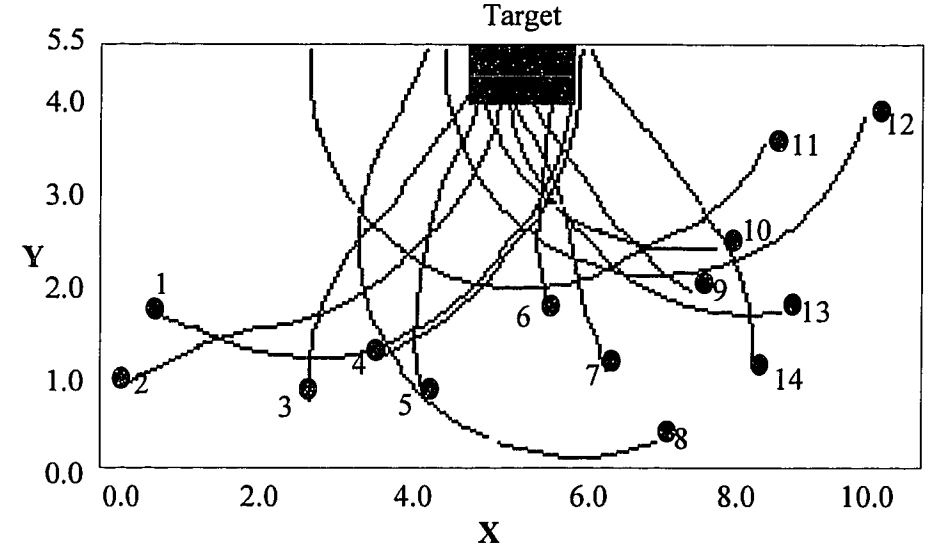
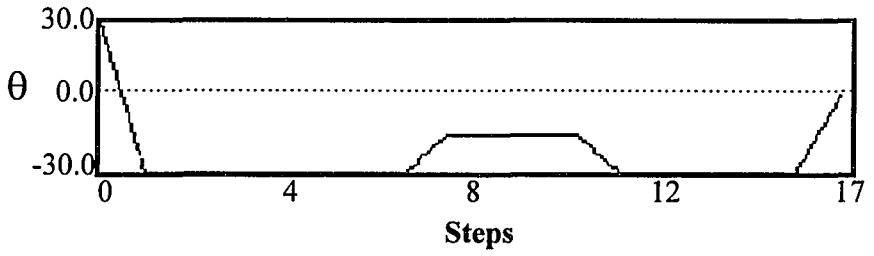
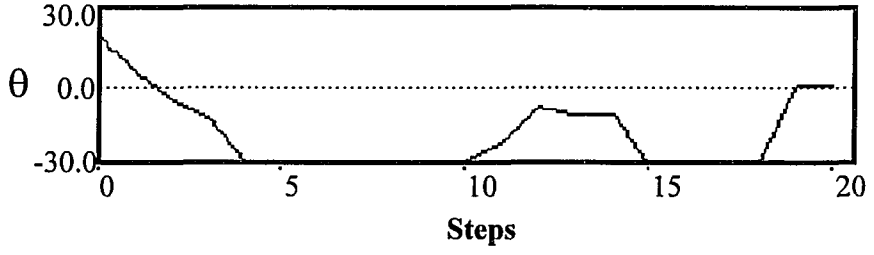


Figure 4.19 Simulated trajectories for the car parking control problem with 14 different locations using a fuzzy controller.



(a)



(b)

Figure 4.20 Control activity of a simulated trajectory (a) using the table-based controller, and (b)using the fuzzy controller.

CHAPTER 5

CONCLUSION

In this dissertation, a cell mapping based systematic method is developed to generate a fuzzy controller for global optimal control of dynamic systems. The systematic method constructs groups of optimal trajectories based on optimal control and path data obtained in terms of cell to cell mapping. This reduces the complexity of constructing a global fuzzy controller compared with generating fuzzy rules based on control data for all the cells in the cell space.

A trajectory group consists of a set of trajectories whose statistical properties are similar. The statistical properties are the mean location, its standard deviation, and the main control level of a trajectory group. Based on these statistical properties, we can create fuzzy membership functions for each trajectory group. To extract the statistical properties of a group, we developed the following procedure:

(1) To derive trajectory, each cell is characterized as an initial, a simple, or a merged cell. A trajectory is a connection of cells from an initial cell to a merged cell via simple cells or from a merged cell to another merged cell via simple cells.

(2) Two features of a trajectory are used as a quantitative measure to determine the similarity of two trajectories: i) Locations of the start and end of the trajectory, and ii) The two highest control levels of the cells in the trajectory. Based on these two features, grouping of similar trajectories is performed using statistical pattern synthesis.

(3) The extracted trajectory groups are used to generate a global fuzzy controller.

A ship navigation problem and a car parking problem are used to demonstrate the applicability of the developed method for 2 and 3 dimensional control problems, respectively. Simulation results show that the performance of the fuzzy controller approximates that of the table-based controller, despite that a small number of rules is used in the fuzzy controller while a data table of a large number of cells is used in the table-based controller.

APPENDIX

```
/* ************************************************************************************/
/*                                                                                    */
/*      Cell-to-Cell Mapping for a Ship Navigation Problem                            */
/*                                                                                    */
/************************************************************************************/
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#define Noc 2
#define Nm 3
#define Non 2
#define Noct 4
#define mfn 16
#define chno 3
#define null (char *) 0
#define PI    (3.14159265)

float hq,hqv,nq,nqv,tsc,xmargin,ymargin,abc1,abc2;
int nt1,idn,unc,n1,n2,ndt,hno,vno,ce2[2], utp,imp,q[2],k[2],type1,chkcell[4];

struct cell{
     int cor[2];
     int cout[2];
     int ctrl[Noct];/*u{1,2,..,16},time interval to i.cell,step #,switchings*/
     int cost;
     int dva;
     int imag;
} cmap ;
struct cell *order[442];
struct cell *temp[500];

struct utr{ int cor[2]; } umap;
struct utr *utrl[1000];
int   utem[800][2];

void wrapping_layer(FILE *fp);
void optimal_cell_mapping(float u[,FILE *fp);
void cellmap(float u[],int ctrl[],int cout[],int cost[],int);
void pointmapping(float u[],float f[],float v[]);
int pdex(float f[]);
int alocel(int ce[],int cot[],int cont[],int cst[],int img,int flg,FILE *fp);
int chos(int npt[],int np);
int ntoq(int nv[],float v[]);
int qton(float v[],int nv[]);
int qtoij(float v[],int nv[]);
int ijtoq(int nv[], float v[]);
int ijton(int ce[]);
int ntoij(int n,int ce[]);
char ** create_matrix(int row_n,int col_n,int element_size);
void free_matrix(char **matrix,int row_n);
int find(int ce[], int);
int found(int ce[], int);

/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/
void wrapping_layer(FILE *fp)
{
int j1,l1,ce[Noc];
char st[Nm];
float u[Noc];
```

```
int wrap,i,j,jj,l,ll,m,mm,n,nn,j2,ce1[200][2];

wrap = (int)(n1/2+0.5);

for(i=0; i<wrap; i++){/* start of #1*/
     k[0]=2*i+1; k[1]=2*i+1;
    j2=(k[0]+2)*(k[1]+2) - k[0]*k[1];
     q[0]=ce2[0]-i; q[1]=ce2[1]-i;

     for(j=0; j<k[0]; j++){ce1[j][0]=q[0]-1; ce1[j][1]=q[1]+j;}
    for(l=0; l<k[0]; l++){ll=k[0]+l; ce1[ll][0]=q[0]+k[1]; ce1[ll][1]=q[1]+l;}
    for(m=0; m<k[1]; m++){ mm=k[0]+k[1]+m; ce1[mm][0]=q[0]+m;    ce1[mm][1]=q[1]-1;}
     for(n=0; n<k[1]; n++){nn=k[0]+k[1]+k[1]+n; ce1[nn][0]=q[0]+n; ce1[nn][1]=q[1]+k[1];}

     ce1[nn+1][0]=q[0]-1; ce1[nn+1][1]=q[1]-1;
    ce1[nn+2][0]=q[0]-1; ce1[nn+2][1]=q[1]+k[1];
     ce1[nn+3][0]=q[0]+k[1]; ce1[nn+3][1]=q[1]+k[1];
     ce1[nn+4][0]=q[0]+k[1]; ce1[nn+4][1]=q[0]-1;
     chkcell[0]=ce1[nn+1][0];/* following four lines are used to*/
     chkcell[1]=ce1[nn+1][1];/* check whether its image cell is*/
     chkcell[2]=ce1[nn+3][0];/* out of wrapiing layer. The celmap() */
     chkcell[3]=ce1[nn+3][1];/* routine is used to and temporal utem[] discarded*/

     for (j1=0; j1<j2;j1++){
          ce[0]=ce1[j1][0]; ce[1]=ce1[j1][1];

          utp=0;
          if(find(ce,-1)>=0) continue;
          if(type1==0)if(found(ce,-1)>=0) continue;
          ntoq(ce,u);
          optimal_cell_mapping(u,fp);
     }
 }
return(0);
}/* of wrapping_layer */
/* ***************************************************************************/
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

optimal_cell_mapping(float u[Noc],FILE *fp)
{
int i,j,k,m,m1,n,np,ftst,strc,l,mu;
int flg[mfn],npt[mfn],ctrl[Noct],cost[2], cot[Noc],cin[Noc],cout[Noc];
float v[Noc];
struct utr *info1;
char st[Nm];

ftst=0;
np=0;
k=0;
ctrl[1]=1;ctrl[3]=0;ctrl[2]=0;/* initial without error*/
     qtoij(u,cin);
      if(find(cin,-1) >=0) return(m);
     ijtoq(cin,u);
     for(i=0;i<16;i++){
          ctrl[0]=i;
          for(n=0;n<Noc;n++) v[n]=u[n];

          if((flg[k++]=celmap(v,ctrl,cout,cost,0))>0) ftst=ftst;
          else{
               if((m1=find(cout,-1))>=0) {
                    cost[0]+=order[m1]->cost;
                    cost[1]+=order[m1]->dva;
                    ctrl[0]=ctrl[0];
                    ctrl[3]=abs((int)(order[m1]->ctrl[0] - ctrl[0]));
                    ctrl[3]=ctrl[3]+order[m1]->ctrl[3];
```

```
                if ((n=alocel(cin,cout,ctrl,cost,m1,-1,fp))>0) printf("Alo.err");
                npt[np++]=imp-1;
            }
            else{
                m=0;
                if(utp==0){
                    for(n=0;n<2;n++) utem[utp][n]=cin[n];
                    utp++;
                }
                else for(n=0;n<utp;n++)
                    if((utem[n][0]==cout[0]) && (utem[n][1]==cout[1])){m=1;break;}
                if(!m){
                    for(n=0;n<2;n++)utem[utp][n]=cout[n];
                    utp++;
                    optimal_cell_mapping(v,fp);
                    if((cin[0]==3 ) && (cin[1]==16)){printf("cin[0]=%4d",cin[0]);}
                    if((m1=find(cout,-1))>=0){
                        cost[0]+=order[m1]->cost;
                        cost[1]+=order[m1]->dva;
                        ctrl[0]=ctrl[0];
                        ctrl[3]=abs((int)(order[m1]->ctrl[0] - ctrl[0]));
                        if ((n=alocel(cin,cout,ctrl,cost,m1,-1,fp))>0) printf("also. error");
                        npt[np++]=imp-1;
                    }
                }
            }
        }
    }

    if(np>0){
        j=chos(npt,np);
        cost[0]=temp[j]->cost;
        cost[1]=temp[j]->dva;
        for(n=0;n< 4;n++) ctrl[n]=temp[j]->ctrl[n];
        for(n=0;n< Noc;n++) cot[n]=temp[j]->cout[n];

        if ((n=alocel(cin,cot,ctrl,cost,temp[j]->imag,1,fp))>0)return(-1);
        for (i=0;i<np;i++) free(temp[--imp]);
        return (0);
    }

        info1=((struct utr *)malloc(sizeof(umap)));
        if(info1==NULL){printf("error in celdtm");exit(1);}
        info1->cor[0]= cin[0]; info1->cor[1]=cin[1];
        utrl[unc]=info1;
        unc++;
 return(flst);
}/* of optimal_cell_mapping */
/* ***************************************************************************/
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

cellmap(float u[Noc],int ctrl[Noct],int cout[Noc],int cost[2],int type)
{
    int i,cin[Noc],l,m,q1;
    int kk=0;
    float v[Noc],f[3],cst,div;
    char st[Nm];
    f[0]=/*1.-(0.1*ctrl[0]); -(2.*ctrl[0]); */(PI*ctrl[0])/8.0;/*-0.3926;*/
    f[1]=kk*1.0;/*ctrl[1]*tsc; 95.11.08*/
    if(idn==25)
        printf("idn==%d\n",idn);
        qton(u,cin);
        if(pmap(u,f,v)<0) return(-1);
        cst=f[2];
```

```
    ch:
     qton(v,cout);
     for(i=0;i<2;i++){
     if(cin[i]!=cout[i]) break;
     if(i==1){
         if((kk++)>4) return(16);
         pointmapping(v,f,v);
         cst+=f[2];
         goto ch;
     }
    }

         f[1]=(kk+1)*tsc;/*(kk+1)*f[1];*/
         for(i=0;i<Noc;i++) u[i]=v[i];
         if((cout[0]>=n1) || (cout[1]>=n2)) return(2);
         if((cout[0]<0) || (cout[1]<0)) return(8);
         if( type ==0){
             if((cout[0]<chkcell[0]) || (cout[0] >chkcell[2]))return(4);/*95.11.12*/
             if((cout[1]<chkcell[1]) || (cout[1] >chkcell[3]))return(4);
         }
          f[2]=cst;
             cost[0]=pdex(f);
         ntoq(cout,u);
         div=fabs((u[0]-v[0])/hq*10.);
         div=div+fabs((u[1]-v[1])/hqv*10.);
         cost[1]=(int)(div);
         for (i=0;i<Noc;i++) u[i]=v[i];
         ctrl[1]=(int)((f[1]/tsc)+/*0.0000001*/0.0);
         return(0);
}/* cellmap */

/* ************************************************************************************************/
/*                                                                                                */
/*    Grouping Optimal Trajectories for a Ship Navigation Problem                                 */
/*                                                                                                */
/**************************************************************************************************/
#define numlevels          16
#define numtraj       21
#define loadn              18
#define Max_Auto_Class  100
#define TOO_MANY_CLASSES  1
int Next_Auto_Class;
#define numfvs   4 /* The # of feature vector *fv */
#define numfv1s  5 /* The # of 2 highest control levels+2 loc.+step*/
#define change_dir 997

typedef struct path{
    int cor[numcells];
    int image;
    int step;
    int ctrl; /* control input */
    int merged;
    int transient;
    int trans_step;
    int pre_check;
    int traject;/* traject which include this cell */
    int group;/* group which include this cell */
    int m_dir;/* moving direction(forward,backward) */
    int flag;
    int fan_in;
}pathmap;
struct path *order[450];

typedef struct trajectory{
    int type;/* 0 is mer-> merge 1 is init->mer */
    int domain;
    int image;
```

```
    int step;
    int level[numlevels];
    int group;/* group which include this trajectory */
}TRAJECTORYS;
struct trajectory *traj[150];
struct class_list_entry {
    int type;
    int ctrl;
    int traj_num; /* the trajectory number */
    float *fv;/* The feature vector Coordinates for he start and end of trajectory int x[numcells],y[],p[]; */
    float *fv1;/* The feature vector Control levels */
    float xmean[numcells];
    float xstd[numcells];
    struct class_list_entry *next;
}GROUPS;
struct class_list_entry *auto_classes[Max_Auto_Class];

typedef struct repretraj{
    int cor[numcells];
    int ctrl; /* control input */
    float xmean[numcells];
    float xstd[numcells];
    int group;
    int traj;
}repretrjmap;
struct repretraj *repre_cell[500];

void check1(int,int,int,int,int);
void sort(int total_n_cells);
void add_information(int, int type);
int compare(int i,int image_cell,int);
int diff(int domain,int image_cell);
void store(int,int,int);
int load_store_group(int);
int load(int);
void add_node(int,int);
void sort1(int);
void connect(int);
int sort2(int,int data[2]);
void add_traj(int,int,int);
void tconnect(int,int,int,int out[loadn]);
float distance_e_n (float *fv1, float *fv2, int n,int *error_code,int);
float *alloc_fv (int n, int *error_code);
double *alloc_dv (int n, int *error_code);
int *alloc_iv (int n, int *error_code);
struct class_list_entry *insert_class(struct class_list_entry **list, float *fv, int n,float *fv1,int n1,int   *error_code,int);
void compute_centroid (struct class_list_entry **list, int n,int,int *error_code,int);
void recompute_centroid (struct class_list_entry **list,int n,int, int *error_code,int);
int classify_auto (struct class_list_entry **list, float *fv,int n, float thr,float *fv1,int,float, int*error_code,int,int);
float *alloc_traj(int,int,int *error_code,int);
int count_traj (struct class_list_entry **list,int,int);
int group_trajectories(int);
void traj_check(void);
void incert_cell(int group,int trajn);
int acount_traj (struct class_list_entry **list,int,int,FILE *fp);
void freestruct(struct class_list_entry **list,int);
int xyptoijk(float v[numcells],int nv[numcells]) ;
int ijktoxyp(int nv[numcells],float v[numcells]);

/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/
void sort(int total_n_cells){
    nt i,image_cell;
    for(i=total_n_cells; i >= 0 ;i--){
        image_cell =order[i]->image;
```

```
        if(order[image_cell]->pre_check>0){
            add_information(image_cell,0);
            continue;
        }
        compare(i,image_cell,0);/* step=0 */
    }
}
/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/

void add_node(int cell,int type){

    if(type ==0) order[cell]->flag = 1;
    if(type ==1) {
        printf("cell %d \n",cell);
        order[cell]->fan_in ++;
    }
}
/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/

void tconnect(int dcell,int ntraj,int type,int out[loadn]){
    int i,j,k,mr,dom_cell,image_cell,level,ctrl[numlevels],flag;

    for(j=0; j<numlevels; j++) ctrl[j]=0;
    i=0;flag =1;

    dom_cell =dcell;
    level = order[dom_cell]->ctrl;

    if (SIMULATION_TYPE==ShipNavigation){
        for(k=0; k<numlevels; k++)
            if(level==k){ctrl[k]++;break;}
    }
    else{
        for(k=0; k<numlevels; k++)
            if(level==(15*k-30)){ctrl[k]++;break;}
    }
    i++;

    while(flag ){
        image_cell=order[dom_cell]->image;
        mr = order[image_cell]->merged;
        order[image_cell]->traject=ntraj;

        if (type==0){/*if the domain cell is a merged cell*/
            if (mr==0 || image_cell==0){
                image_cell = dom_cell;flag =0;
            }
            else{
                level = order[image_cell]->ctrl;
                if (SIMULATION_TYPE==0){
                    for(k=0; k<numlevels; k++)
                    if(level==k){ctrl[k]++;break;}
                }
                else{
                    for(k=0; k<numlevels; k++)
                    if(level==(15*k-30)){ctrl[k]++;break;}
                }
                i++;
                /* this line for sort2() not duplicate*/
                order[image_cell]->m_dir=change_dir;
                dom_cell =image_cell;
```

```
            }
        }
        else{/*if the domain cell is an initial cell*/
            if (mr!=0 || image_cell==0){
                image_cell = dom_cell;flag =0;
            }
            else{
                level = order[image_cell]->ctrl;

                if (SIMULATION_TYPE==0){
                    for(k=0; k<numlevels; k++)
                    if(level==k){ctrl[k]++;break;}
                }
                else {
                    for(k=0; k<numlevels; k++)
                    if(level==(15*k-30)){ctrl[k]++;break;}
                }
                i++;
                dom_cell =image_cell;
            }
        }
    }

    out[0]= image_cell;/* image cell which locate before the merged cell */
    out[1]= i;/*steps from domain cell to image cell of a trajectory */
    for(j=2; j<loadn; j++) out[j]=ctrl[j-2];
    /*accumulated control levels for similarity checking*/
}

/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/

/* Create a new class list entry for the given feature vector nd insert it into the singly linked list LIST. Return a
    pointer to the new entry */

struct class_list_entry *insert_class (struct class_list_entry **list, float *fv, int n,
    float *fv1, int n1,int *error_code,int trajn)
{
    int i;
    struct class_list_entry *x;
    float *fv2,*fv3;

    *error_code = 0;
    x = (struct class_list_entry *)
        malloc(sizeof(struct class_list_entry));
    if (x == 0) {
        *error_code = OUT_OF_STORAGE;
        return 0;
    }

    fv2 = alloc_fv(numfvs, error_code);
    if (*error_code) return 0;
    for (i=0; i<n; i++) fv2[i] = fv[i];

    fv3 = alloc_fv(numfv1s, error_code);
    if (*error_code) return 0;
    for (i=0; i<n1; i++) fv3[i] = fv1[i];

    x->fv = fv2;
    x->fv1 = fv3; x->traj_num=trajn; x->type=traj[trajn]->type;
    x->next = 0;
    for(i=0; i<numcells; i++)x->xmean[i]=-999.0;
    for(i=0; i<numcells; i++)x->xstd[i]=-999.0;
    x->ctrl = -999;
    if (*list == 0) *list = x;
```

```
    else {
        x->next = *list;
        *list = x;
    }
    return x;
}
/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/

/*Automatic clustering based on the threshold distance THR. */
int classify_auto (struct class_list_entry **list, float *fv,int n, float thr,float *fv1,int n1,float thr1,int *error_code,int trajn,int type)
{
    float *z,*z1, dsave, d, d1,d2,d3;
    struct class_list_entry *x;
    int i, class;

    *error_code = 0;
    z = alloc_fv (n, error_code);
    if (*error_code) return -1;
    z1 = alloc_fv (n1, error_code);
    if (*error_code) return -1;
    for (i=0; i<n; i++) z[i] = fv[i];
    for (i=0; i<n1; i++) z1[i] = fv1[i];

    if (list[0] == 0) {/* Empty list; insert FV */
      x = insert_class
        (&(list[0]), z, n,z1, n1, error_code,trajn);
      Next_Auto_Class = 1;
      compute_centroid (&(list[0]), n,n1, error_code,trajn);
      if (*error_code) return -1;
      return 0;
    } else {
      class = -1;      dsave = 1.0e15;
      for(i=0; i<Next_Auto_Class; i++){
        x = list[i];
        /* following line is for matching types of traj */
        if(traj[trajn]->type !=x->type)continue;

        d = distance_e_n (fv, x->fv, n, error_code,0);
        d1 = distance_e_n (fv1, x->fv1, n1, error_code,1);

        if (d < thr) {
            /*d1 = distance_e_n (fv1, x->fv1, n1, error_code,1); */
            if (d1 < dsave) { /* Smallest so far? */
                dsave = d1;
                class = i;
            }
        }
      }

      if (dsave > thr1) { /* Create a new class */
        insert_class (&(list[Next_Auto_Class]),z,n,z1,n1, error_code,trajn);
        if (*error_code) return -1;
        compute_centroid (&(list[0]), n,n1, error_code,trajn);
        if (*error_code) return -1;
        if (Next_Auto_Class > Max_Auto_Class) {
            *error_code = TOO_MANY_CLASSES;
            return -1;
        }
        class = Next_Auto_Class;
        Next_Auto_Class++;
      } else {
        insert_class (&(list[class]), z,n,z1,n1,error_code,trajn);
        if (*error_code) return -1;
        recompute_centroid (&(list[class]), n,n1,error_code,trajn);
```

```
            if (*error_code) return -1;
          }
        }
        return class;
}

/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/

int group_trajectories(int total_trajectory){
    FILE *fp3;
    int i,j,k=0,sum,error_code,ii,jj,suml,l;
    float *fv,*fv1,thr,thr1;

    for(ii=0;ii<1; ii++){
        for(jj=0;jj<1; jj++){
            printf("Start of autoclass i=%3d,j=%3d,thr=%4.3f thr1=%4.3f\n",ii,jj,thr,thr1);
            for(i=0; i<total_trajectory; i++){

            fv = alloc_traj(numfvs,i,&error_code,0);
            if (error_code)    {printf("Error in class_t1\n");return -1;}
            fv1 = alloc_traj(numfv1s,i,&error_code,1);
            if (error_code)    {printf("Error in class_t2\n");return -1;}

            classify_auto (auto_classes,fv,numfvs,thr,fv1,numfv1s,thr1,&error_code,i,i);
            if (error_code)    {printf("Error in class_t3\n");return -1;}
        }
        printf("Next_Auto_Class=%d ",   Next_Auto_Class);
        sum=0;suml=0; j=0;l=0;

    for(i=0; i<Next_Auto_Class; i++){
        if((j=count_traj (auto_classes,i,0))>=0){
            k++;
            if(j>=1) l=acount_traj(auto_classes,i,1,fp3);
        }
        sum=sum+j; suml=suml+l;
    }
    fclose(fp3);
    }
}

/* ************************************************************************************/
/*                                                                                    */
/* Generating Fuzzy Rule Base for a Ship Navigation Problem                           */
/*                                                                                    */
/**************************************************************************************/
struct fuzmemb{
    float fmin,fmax;/*global universe of intercourse */
    float fmean; /* centroid of cell group regions */
    float fsig;  /* standard deviation from centroid of groups of cells */
};

typedef unsigned char UBYTE;
typedef signed char SBYTE;
typedef UBYTE FUBYTE;
typedef signed long SLONG;
typedef SBYTE FSBYTE;

int *Triangular(struct fuzmemb *fx);
int *Trapozoidal(struct fuzmemb *fx);
void GenerateFuzzyRulebase(void);
int ** create_imatrix(int row_n,int col_n,int element_size);
void ** free_imatrix(int **matrix,int row_n);
int **fuzzy_set_matrix;
int *fuzzy_rule_matrix;
```

```
int load_store_fuzzyset(int,int);
int load_store_cellset(int type);

/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/

void GenerateFuzzyRulebase(void)
{
    FILE *fp;
    int i,j,m1,m2,k,error_code,*tmp,flag;
    /* m1 is the maximum possible number of groups for fuzzy sets */
    struct fuzmemb *fx;
    float xmean,xstd,Ymax,Ymin,Xmax,Xmin,Pmax,Pmin;

    if (SIMULATION_TYPE == ShipNavigation){
        Ymax = 1.05;
        Ymin = -1.05;
        Xmax = 1.05;
        Xmin = -1.05;
        Pmax = 4.712389;
        Pmin =-1.5707963;
    } else {
        Ymax = 0.4*Total_Celly;
        Ymin = 0.0;
        Xmax = 0.4*Total_Cellx;
        Xmin = 0.0;
        Pmax = 4.712389;
        Pmin =-1.5707963;
    }
    flag=0;

    if (Next_Auto_Class ==0){
        m1=load_store_group(1);Next_Auto_Class=m1;flag=1;
    }   else {m1=check_group();}
    m2=m1;
    if(fuzzy_set_matrix!=NULL)  free_imatrix(fuzzy_set_matrix,numcells*m2);
    if(fuzzy_rule_matrix!=NULL) free(fuzzy_rule_matrix);
    fuzzy_set_matrix=(int **)create_imatrix(numcells*m2,256,sizeof(int));
    error_code=0;
    fuzzy_rule_matrix=alloc_iv(m2,&error_code);
    if (error_code)    {printf("Error in class_t12\n");exit(1);}

    m1=0;
    for(i=0; i<Next_Auto_Class;i++){
        if(auto_classes[i]->xmean[0]!=-999.0){
            for(j=0;j<numcells;j++){

                if((fx = (struct fuzmemb *)malloc(sizeof(struct fuzmemb)))==0)
                    {printf("error in generate_fuzzy_rulebase()"); exit(0);}
                    xmean=auto_classes[i]->xmean[j];
                    xstd=auto_classes[i]->xstd[j];

                if (SIMULATION_TYPE== ShipNavigation){
                    if(j==0){fx->fmax=Xmax;fx->fmin=Xmin;
                    } else{   fx->fmax=Ymax;fx->fmin=Ymin; }
                }else {
                    if(j==0){fx->fmax=Xmax;fx->fmin=Xmin;
                    } else if(j==1){fx->fmax=Pmax;fx->fmin=Pmin;
                    } else{   fx->fmax=Ymax;fx->fmin=Ymin; }
                }
                fx->fmean=xmean;fx->fsig=xstd;

                tmp=Trapozoidal(fx);
                for(k=0;k<256;k++)fuzzy_set_matrix[numcells*m1+j][k]=tmp[k];
            }
```

```
            /*auto_classes[group]->fv1[2] is the highest control level used for
                the representative trajectory*/
            if(flag==1)fuzzy_rule_matrix[m1]=auto_classes[i]->ctrl;
            else fuzzy_rule_matrix[m1]=(int)(auto_classes[i]->fv1[2]);
        m1++;
        }
    }
load_store_fuzzyset(m2,0);
}
/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/

int ** create_imatrix(int row_n,int col_n,int element_size)
{
    int **matrix,i;

    matrix=(int **)malloc(sizeof(int *)*row_n);
    if(matrix==NULL){   printf("error in create_imatrix\n"); exit(1);}

    for(i=0;i<row_n;i++){
        matrix[i]=(int *)malloc(element_size*col_n);
        if(matrix[i]==NULL){      printf("error in create_imatrix\n"); exit(1);}
    }
    return(matrix);
}
/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/

int *Triangular(struct fuzmemb *fx)
{
/* Generate 8 bits fuzzy membership functions (0 to 255) based on given geometrical coordinates of the cells ofa group. */
    int  i,imaxx,iminx,imbf[3],numx[2],iouty,error_code=0;
    float fmaxx,fminx,fscalex,slope[2];
        float fLeft,fApex,fLeft_Apex,fRight_Apex,fRight;/* data of region */
        int *temp;

        iminx=0; imaxx=255+1;
        fmaxx=fx->fmax;
        fminx=fx->fmin;
        fLeft=fx->fmean - (fx->fsig)*(7./5.);
        fApex=fx->fmean;
        fRight=fx->fmean +(fx->fsig)*(7./5.);
        if(fLeft >= fmaxx) {printf("error in *triangular \n"); exit(1);}
        if(fRight <= fminx) {printf("error in *triangular \n"); exit(1);}
        if(fLeft <= fminx) fLeft=fminx;
        if(fRight>= fmaxx) fRight=fmaxx;

        temp=alloc_iv(imaxx,&error_code);
        if (error_code) return 0;

        fscalex=(imaxx - iminx)/(fmaxx-fminx);
        imbf[0]=(int)((fLeft-fminx)*fscalex);
        numx[0]=(int)((fApex - fLeft)*fscalex);
        imbf[1]=imbf[0]+numx[0];
        numx[1]=(int)((fRight - fApex)*fscalex);
        imbf[2]=imbf[1]+numx[1];

        for(i=0; i<imaxx; i++){
            if((i>=0) && ( i < imbf[0])){temp[i]=0; continue;}

            if((i >= imbf[0]) && (i < imbf[1])){
                slope[0]=255./((fApex-fLeft)*fscalex);
                iouty = (int)(slope[0]*(i -(fLeft*fscalex)));
```

```
                if(iouty <0) temp[i]=0;
                else if(iouty>255) temp[i]=255;
                else temp[i]=iouty;
                continue;
            }
            else if((i >= imbf[1]) && (i < imbf[2])){
                slope[1]=-255./((fRight-fApex)*fscalex);
                iouty = (int)(slope[1]*(i -(fApex*fscalex))) +255.;
                if(iouty <0) temp[i]=0;
                else if(iouty>255) temp[i]=255;
                else temp[i]=iouty;
                continue;
            }
            else if((i>=imbf[2]) && ( i < 256))temp[i]=0;
            }
    return(temp);
}/* of *Triangular */
/* ************************************************************************/
/*                                                                        */
/*                                                                        */
/**************************************************************************/

int *Trapozoidal(struct fuzmemb *fx)
{
/* Generate 8 bits fuzzy membership functions (0 to 255) based on given geometrical coordinates of the cells of a group.*/
        int  i,imaxx,iminx,imbf[4],numx[3],iouty,error_code=0;
        float fmaxx,fminx,fscalex,slope[2];/*global data */
        float fLeft,fLApex,fRApex,fRight;
        float fLeft_Apex,fRight_Apex;/* data of region */
        int *temp;

        iminx=0; imaxx=255+1;
        fmaxx=fx->fmax;
        fminx=fx->fmin;
        fLeft    =fx->fmean- (fx->fsig)*(6.0/5.);
        fLApex=fx->fmean- (fx->fsig)*(3.5/5.);
        fRApex=fx->fmean+ (fx->fsig)*(3.5/5.);
        fRight=fx->fmean+ (fx->fsig)*(6.0/5.);
        if(fLeft >= fmaxx) {printf("error in *triangular \n"); exit(1);}
        if(fRight <= fminx) {printf("error in *triangular \n"); exit(1);}
        if(fLeft  <= fminx) fLeft=fminx;
        if(fRight >= fmaxx) fRight=fmaxx;
        if(fLApex <= fminx) fLApex=fminx;
        if(fRApex >= fmaxx) fRApex=fmaxx;

        temp=alloc_iv(imaxx,&error_code);
        if (error_code) {exit(1);}
        fscalex=(imaxx - iminx)/(fmaxx-fminx);
        imbf[0]=(int)((fLeft-fminx)*fscalex);
        numx[0]=(int)((fLApex - fLeft)*fscalex);
        imbf[1]=imbf[0]+numx[0];
        numx[1]=(int)((fRApex - fLApex)*fscalex);
        imbf[2]=imbf[1]+numx[1];
        numx[2]=(int)((fRight - fRApex)*fscalex);
        imbf[3]=imbf[2]+numx[2];

        for(i=0; i<imaxx; i++){
            if((i>=0) && ( i < imbf[0])){temp[i]=0; continue;}
            if((i >= imbf[0]) && (i < imbf[1])){
                slope[0]=255./((fLApex-fLeft)*fscalex);
                iouty = (int)(slope[0]*(i -(fLeft*fscalex)));
                if(iouty <0) temp[i]=0;
                else if(iouty>255) temp[i]=255;
                else temp[i]=iouty;
                continue;
            }
            else if((i >= imbf[1]) && (i < imbf[2]))temp[i]=255;
```

```
            else if((i >= imbf[2]) && (i < imbf[3])){
                slope[1]=-255./((fRight-fRApex)*fscalex);
                iouty = (int)(slope[1]*(i -(fRApex*fscalex))) +255.;
                if(iouty <0) temp[i]=0;
                else if(iouty>255) temp[i]=255;
                else temp[i]=iouty;
                continue;
            }

        else if((i>=imbf[3]) && ( i < 256))temp[i]=0;
        }
return(temp);
}/* *Trapozoidal */

/* ************************************************************************************/
/*                                                                                    */
/*      Global Fuzzy Controller for a Ship Navigation                                 */
/*                                                                                    */
/**************************************************************************************/

struct FCCicb _SteerTheta_NB_cbinfo = { -3076L, 28L };
struct FCCicb _SteerTheta_NM_cbinfo = { -2752L, 43L };
struct FCCicb _SteerTheta_NS_cbinfo = { -841L,  29L };
struct FCCicb _SteerTheta_ZE_cbinfo = { 0L, 21L };
struct FCCicb _SteerTheta_PS_cbinfo = { 841L, 29L };
struct FCCicb _SteerTheta_PM_cbinfo = { 2752L, 43L };
struct FCCicb _SteerTheta_PB_cbinfo = { 3076L, 28L };

float gobal_fuzzy_controller(float xi[numcells],int,int cc[2]);
long SteerThetaf(int i, int Link,int);

/* ************************************************************************************/
/*                                                                                    */
/*                                                                                    */
/**************************************************************************************/

float gobal_fuzzy_controller (float xi[numcells],int m2,int contnum[2]){
    float Thetaf;
    int i,j,inprule,fzset,fzloc;
    long int rvalue;
    SBYTE SteerTheta;
    int _XYP1[numcells];
    struct FCCicb _SteerTheta_temp;
    double maxval,minval,_XYP[numcells];
    int maxindex,minindex,Max_fuzzy_set;
    float XcoordScaleFactor,YcoordScaleFactor,PHIcoordScaleFactor;
    float DesiredXcoordf,DesiredYcoordf,DesiredPHIcoordf;

    memset (&_SteerTheta_temp, 0, sizeof(_SteerTheta_temp));

    if (SIMULATION_TYPE==ShipNavigation){
        XcoordScaleFactor=(256./2.1);
        YcoordScaleFactor=(256./2.1);
        DesiredXcoordf = 0.0;
        DesiredYcoordf = 0.0;
        _XYP1[0]=(int) ((xi[0]+1.05)*XcoordScaleFactor);
        _XYP1[1]=(int) ((xi[1]+1.05)*YcoordScaleFactor);
        if (_XYP1[0] <= 0)    _XYP1[0] = 0;
        else if (_XYP1[0] >= 256)_XYP1[0] = 256;
        if (_XYP1[1] <= 0)    _XYP1[1] = 0;
        else if (_XYP1[1] >= 256)_XYP1[1] = 256;
    } else{
        XcoordScaleFactor=(256./11.5);
        YcoordScaleFactor= (256./5.5);
        PHIcoordScaleFactor=(256./(2*3.14159));
        DesiredXcoordf = 5.5;
        DesiredYcoordf =5.5;
```

```
        DesiredPHIcoordf = 1.57;
        _XYP1[0]=(int) ((xi[0])*XcoordScaleFactor);
        _XYP1[2]=(int) ((xi[2])*YcoordScaleFactor);
        _XYP1[1]=(int) ((xi[1]+1.58)*PHIcoordScaleFactor);
        if (_XYP1[0] <= 0)   _XYP1[0] = 0;
        else if (_XYP1[0] >= 256)_XYP1[0] = 256;
        if (_XYP1[1] <= 0)   _XYP1[1] = 0;
        else if (_XYP1[1] >= 256)_XYP1[1] = 256;
        if (_XYP1[2] <= 0)   _XYP1[2] = 0;
        else if (_XYP1[2] >= 256)_XYP1[2]= 255;
    }
    contnum[0]=0;
    if(Next_Auto_Class==0)Max_fuzzy_set=m2;
    else Max_fuzzy_set=check_group();

    for(j=0; j<Max_fuzzy_set; j++){
            for(i=0;i<numcells;i++){
                    fzset=fuzzy_set_matrix[numcells*j+i][_XYP1[i]];
                    _XYP[i]=(double)(fzset*1.0);
            }
            MaxMin1D(_XYP,numcells,&maxval,&maxindex,&minval,&minindex);
            inprule=fuzzy_rule_matrix[j];

            if ( minval!=0.0){
                if (SIMULATION_TYPE== ShipNavigation){
                    rvalue=SteerThetaf(3,inprule,0);
                    _SteerTheta_temp.moment += minval*rvalue;
                    _SteerTheta_temp.area += minval*SteerThetaf(4,inprule,0);
                } else {
                    rvalue=SteerThetaf(1,inprule,1);
                    _SteerTheta_temp.moment += minval*rvalue;
                    _SteerTheta_temp.area += minval*SteerThetaf(2,inprule,1);
                }
                contnum[0]++;
            }
    }
    if (_SteerTheta_temp.area != 0L)
        SteerTheta = ((SBYTE) (_SteerTheta_temp.moment / _SteerTheta_temp.area));
    else
        SteerTheta= ((SBYTE) 0);

    if (SIMULATION_TYPE== ShipNavigation){
        Thetaf =(SteerTheta*2.*PI)/16.;
        if(Thetaf>=2*PI)Thetaf=2.*PI;
        if(Thetaf<=0.) Thetaf=0.0;
    } else {
        Thetaf =(SteerTheta*35.)/127.;
        if(Thetaf>=30.0)Thetaf=30.0;
        if(Thetaf<=-30.) Thetaf=-30.0;
    }
return(Thetaf);
}/* of gobalFuzzyController */

/* ****************************************************************************/
/*                                                                            */
/*                                                                            */
/******************************************************************************/

long SteerThetaf(int i, int Link,int type)
{
long int value;
    if(type==1){
        switch(Link){
            case 0:Link=0;break;
            case 1:Link=1;break;
            case 2:Link=3;break;
            case 3:Link=5;break;
```

```
            case 4:Link=6;break;
            default:break;
        }
    }
    if(i==1){
        switch(Link){
            case 0: value=_SteerTheta_NB_cbinfo.moment;break;
            case 1: value=_SteerTheta_NM_cbinfo.moment;break;
            case 2: value=_SteerTheta_NS_cbinfo.moment;break;
            case 3: value=_SteerTheta_ZE_cbinfo.moment;break;
            case 4: value=_SteerTheta_PS_cbinfo.moment;break;
            case 5: value=_SteerTheta_PM_cbinfo.moment;break;
            case 6: value=_SteerTheta_PB_cbinfo.moment;break;
            default:break;
        }
    } else if(i==2){
        switch(Link){
            case 0: value=_SteerTheta_NB_cbinfo.area;break;
            case 1: value=_SteerTheta_NM_cbinfo.area;break;
            case 2: value=_SteerTheta_NS_cbinfo.area;break;
            case 3: value=_SteerTheta_ZE_cbinfo.area;break;
            case 4: value=_SteerTheta_PS_cbinfo.area;break;
            case 5: value=_SteerTheta_PM_cbinfo.area;break;
            case 6: value=_SteerTheta_PB_cbinfo.area;break;
            default:break;
        }
    } else if(i==3 ) {
        switch(Link){
            case 0: value=_SteerAngle_00.moment;break;
            case 1: value=_SteerAngle_01.moment;break;
            case 2: value=_SteerAngle_02.moment;break;
            case 3: value=_SteerAngle_03.moment;break;
            case 4: value=_SteerAngle_04.moment;break;
            case 5: value=_SteerAngle_05.moment;break;
            case 6: value=_SteerAngle_06.moment;break;
            case 7: value=_SteerAngle_07.moment;break;
            case 8: value=_SteerAngle_08.moment;break;
            case 9: value=_SteerAngle_09.moment;break;
            case 10: value=_SteerAngle_10.moment;break;
            case 11: value=_SteerAngle_11.moment;break;
            case 12: value=_SteerAngle_12.moment;break;
            case 13: value=_SteerAngle_13.moment;break;
            case 14: value=_SteerAngle_14.moment;break;
            case 15: value=_SteerAngle_15.moment;break;
        }
    } else {
        switch(Link){
            case 0: value=_SteerAngle_00.area;break;
            case 1: value=_SteerAngle_01.area;break;
            case 2: value=_SteerAngle_02.area;break;
            case 3: value=_SteerAngle_03.area;break;
            case 4: value=_SteerAngle_04.area;break;
            case 5: value=_SteerAngle_05.area;break;
            case 6: value=_SteerAngle_06.area;break;
            case 7: value=_SteerAngle_07.area;break;
            case 8: value=_SteerAngle_08.area;break;
            case 9: value=_SteerAngle_09.area;break;
            case 10: value=_SteerAngle_10.area;break;
            case 11: value=_SteerAngle_11.area;break;
            case 12: value=_SteerAngle_12.area;break;
            case 13: value=_SteerAngle_13.area;break;
            case 14: value=_SteerAngle_14.area;break;
            case 15: value=_SteerAngle_15.area;break;
        }
    }
return(value);
}/* SteerThetaf */
```

REFERENCES

[1] C. S. Hsu, "A Theory of cell-to-cell mapping dynamics systems," *J. of Appl. Mech.*, vol. 47, 1980.

[2] C. S. Hsu, "A discrete method of optimal control based upon the cell state space concept," *J. of Appl. Mech.*, vol. 46, pp. 547-569,1985.

[3] W. H. Zhu, "An applied cell mapping method for optimal control systems," *J. of Opt. Theo. Appl.*, vol. 60, no. 3, 1989.

[4] W. H. Zhu and M. C. Leu, "Planning optimal robot trajectories by cell mapping," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1990.

[5] S. H. Johnson, D. G. Harlow, and J. S. Yoon, "Time-optimal multistage controllers for nonlinear continuous processes," *Trans. of the ASME*, vol. 108, 1986.

[6] F. H. Bursal, and C. S. Hsu, "Application of a cell-mapping method to optimal control problems," *Int. J. of Control*, vol. 49, 1989.

[7] Y. Y. Chen and T. C. Tsao, "A Description of Dynamical Behavior of Fuzzy Systems," *IEEE Trans. Syst, Man, Cybern.*, vol. 19, no. 4, pp. 745-755, 1989.

[8] S. M. Smith and D. J. Comer, "Automated calibration of a fuzzy logic controller using a cell state space algorithm," *IEEE Control Systems*, pp. 18-28, Aug., 1991

[9] H. Kang and G. V. Vachtsevanos, "Nonlinear Fuzzy based on the vector fields of the phase portrait assignment algorithm," *Proc. ACC* '90., 1990.

[10] T. Takagi and M. Sugeno, "Fuzzy identification of systems and application to modeling and control," *IEEE Trans. Syst., Man, Cybern.*, vol. 15, pp. 116-132, 1985.

[11] J-S. J. Roger, "ANFIS: Adaptive-network-based fuzzy inference systems," *IEEE Trans. Syst., Man, Cybern.*, 1992.

[12] B. H. Tongue, "On obtaining global nonlinear system characteristics through interpolated cell mapping," *Physica*, vol. 28D, 1987.

[13] B. Kosko, *Neural networks and fuzzy systems*. Prentice Hall, Englewood Cliffs, NJ, 1992.

REFERENCES
(Continued)

[14] L. X. Wang and J. M. Mendel, "Generating fuzzy rules from numerical data from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 6, pp. 1414-1427, 1992.

[15] C. T. Lin and C. S. G. Lee, "Neural-network based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol. 40, no. 12, pp. 1320-1336, 1991.

[16] M. Sugeno and T. Yasukawa. "A fuzzy-logic-based approach to qualitative modeling," *IEEE Trans. Fuzzy Systems.*, vol. 1, no. 1, pp. 7-31, 1993.

[17] S. Abe and M.-S. Lan, "A method for fuzzy rules extraction directly from numerical data and its application to pattern classification," *IEEE Trans. Fuzzy Systems.*, vol. 3, no. 1, pp. 18-28, 1995.

[18] C. L. Karr and E. J. Gentry, "Fuzzy control of pH using genetic algorithms," *IEEE Trans. Fuzzy systems.*, vol. 1, pp. 46-53, 1993.

[19] U. T. Varsek and B. Filipic, "Genetic algorithms in controller design and tuning," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 5, pp. 1330-1338, 1993.

[20] D. DrianKov, H. Hellendoorn, and M. Reinfrank, *An Introduction to Fuzzy Control,* Springer-Verlag, New York, NY,1993.

[21] J.C. Latombe, *Robot Motion Planning* , Kluwer Academic Publishers, Norwell, Massachusetts,1993.

[22] G. F. Luger and W. A. Stubblefield, *Artificial Intelligence and the Design of Expert Systems*, The Benjamin/Cummings Publishing Company, Redwood City, CA, 1989.

[23] H. Wong and M. C. Leu, "Adaptive genetic algorithm for optimal printed circuit board assembly planning," Annals of the CIRP, vol. 42, pp. 17-20, 1993.

[24] S. Chang, *Adaptive Nonlinear Control Using Fuzzy Logic and Neural Networks*, Ph. D. Dissertation, New Jersey Institute of Technology, Newark, New Jersey, 1994.